NAVAL POSTGRADUATE SCHOOL
Monterey, California

DISSERTATION

AN EFFICIENT MODEL-BASED IMAGE UNDERSTANDING METHOD FOR AN AUTONOMOUS VEHICLE

by

Khaled Ahmed Morsy

September 1997

Dissertation Supervisor: Dr. Yutaka Kanayama

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
January 1998

3. REPORT TYPE AND DATES COVERED
Doctoral Dissertation

4. TITLE AND SUBTITLE
AN EFFICIENT MODEL-BASED IMAGE UNDERSTANDING METHOD FOR AN AUTONOMOUS VEHICLE

5. FUNDING NUMBERS

6. AUTHOR(S)
Morsy, Khaled Ahmed

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem discussed in this dissertation is the development of an efficient method for visual navigation of autonomous vehicles.The approach is to significantly reduce the expensive computational time of landmark detection by straight-edge features. A novel, fast straight-edge-detection method for use in autonomous vehicle navigation and other image-understanding applications is presented. Straight edges in gray-scale images are detected using a new direction-controlled edge tracking method, which gives precise estimate of the endpoints. To significantly reduce the number of exhaustive pixel computations, a random-hitting method using a pseudo-random number generator is proposed. Only if a generated pixel is significant do we start tracking the edge containing that pixel. To overcome the "noisy" gradient direction information, a robust least-squares linear fitting method is used to control the tracking process.

The results of the algorithm show how it is efficient for landmark detection, which is important for motion control of autonomous vehicles. Thus the new method is implemented as a component of the image-understanding system in the autonomous mobile robot Yamabico-11 at the Naval Postgraduate School.

An efficient world-modeling method based on the 2D model of the environment of the vehicle, including the heights of vertical edges in the environment, is presented. This modeling method is implemented with the new edge-detection method to improve the efficiency of the pose-determination algorithm (pose is a combination of the position and orientation of the camera), which is an essential task in the area of autonomous vehicle navigation.

14. SUBJECT TERMS
Image understanding, computer vision, edge detection, efficient algorithm,robotics, autonomous vehicle, model-based vision.

15. NUMBER OF PAGES
167

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

AN EFFICIENT MODEL-BASED IMAGE UNDERSTANDING METHOD FOR AN AUTONOMOUS VEHICLE

Khaled Ahmed Morsy
Major, Egyptian Air Force
B.S., Military Technical College, Egypt, 1985
M.S., Military Technical College, Egypt, 1990

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

Author: Khaled A. Morsy
Khaled Ahmed Morsy

Approved by:

Yutaka Kanayama
Professor of Computer Science
Dissertation Supervisor

Xiaoping Yun
Associate Professor of Electrical and Computer Engineering

C. Thomas Wu
Associate Professor of Computer Science

Craig Rasmussen
Associate Professor of Mathematics

Lynne Grewe
Professor of Communication Science, CSU, Monterey Bay
Visiting Associate Professor of Computer Science

Approved by: Ted Lewis
Ted Lewis, Chairman, Department of Computer Science

Approved by: Maurice D. Weir
Maurice D. Weir, Associate Provost for Instruction

ABSTRACT

The problem discussed in this dissertation is the development of an efficient method for visual navigation of autonomous vehicles. The approach is to significantly reduce the expensive computational time of landmark detection by straight-edge features. A novel, fast straight-edge-detection method for use in autonomous vehicle navigation and other image-understanding applications is presented. Straight edges in gray-scale images are detected using a new *direction-controlled edge tracking method*, which gives precise estimate of the endpoints. To significantly reduce the number of exhaustive pixel computations, a *random-hitting method* using a pseudo-random number generator is proposed. Only if a generated pixel is significant do we start tracking the edge containing that pixel. To overcome the "noisy" gradient direction information, a robust least-squares linear fitting method is used to control the tracking process.

The results of the algorithm show how it is efficient for landmark detection, which is important for motion control of autonomous vehicles. Thus the new method is implemented as a component of the image-understanding system in the autonomous mobile robot Yamabico-11 at the Naval Postgraduate School.

An efficient world-modeling method based on the 2D model of the environment of the vehicle, including the heights of vertical edges in the environment, is presented. This modeling method is implemented with the new edge-detection method to improve the efficiency of the pose-determination algorithm (pose is a combination of the position and orientation of the camera), which is an essential task in the area of autonomous vehicle navigation.

TABLE OF CONTENTS

I. INTRODUCTION . . . 1

A. BACKGROUND . . . 1

B. IMAGE UNDERSTANDING . . . 2

1. Feature Extraction . . . 3
2. Segmentation . . . 4
3. Classification and Interpretation . . . 5

C. AUTONOMOUS VEHICLES AND VISION . . . 5

1. Vision-Guided Navigation . . . 6

D. PROBLEM STATEMENTS . . . 8

1. Problem Definition . . . 8
2. Assumptions . . . 9

E. PREVIOUS WORK . . . 9

1. Edge Detection . . . 9
2. Pose Determination . . . 11

F. ORGANIZATION OF DISSERTATION . . . 12

II. PRINCIPLE OF STRAIGHT-EDGE DETECTION USING GRADIENTS . . . 13

A. IMAGE REPRESENTATION . . . 13

B. EDGE DETECTION PROBLEM . . . 15

C. GRADIENT DIRECTIONS AND GRADIENT REGIONS . . . 16

1. Gradient Computation . . . 16
2. Gradient Regions . . . 18

D. FINDING EDGE FEATURES BY LEAST-SQUARES FITTING 21

E. SUMMARY . . . 24

III. STRAIGHT-EDGE-FINDING METHOD BY SCANNING . . . 27

A. IMAGE SCANNING . . . 27

B. CONNECTIVITY TEST 28
C. DATA STRUCTURES 31
1. Row of Pixels 31
2. Gradient Regions 32
3. Line Segment 33
D. ALGORITHM 34
1. Computing Gradient Magnitude 35
2. Connectivity Test Function 36
3. Least-Squares Fitting 36
4. Computing Endpoints of Segments 38
E. EXPERIMENTAL RESULTS 38
IV. DIRECTION-CONTROLLED EDGE TRACKING METHOD 41
A. PRINCIPLE 41
1. Edge Tracking Algorithm 42
B. TRACKING CONTROL BY EDGE DIRECTION 43
1. Tracking Direction Evaluation 44
2. One-Way Pixel Sequence Tracking Algorithm 45
C. SELECTING NEXT PIXEL 45
1. Quantization of Edge Direction 45
2. Next Pixel Selection 46
D. COMPUTING SEGMENT PARAMETERS 48
V. A GLOBAL ALGORITHM FOR EDGE DETECTION USING RANDOM HITTING 51
A. RANDOM HITTING 52
1. Concept of Random Hitting 52
2. Edge Hitting Probability 53
B. CLOSENESS TEST 57
C. RESULTS 60

D. SUMMARY 66
VI. WORLD MODEL 69
A. BACKGROUND 69
B. 2D POLYGONAL WORLD 70
1. Polygons 70
a. General Definitions 70
b. Data Structures 71
2. World 72
a. World Data Structures 73
b. Yamabico's 2D World 73
C. EXTENDING 2D MODEL TO 3D 74
D. $2\frac{1}{2}$D MODEL 76
E. TRANSFORMING MODEL INTO 2D VIEW 78
F. MODELING RESULTS 81
VII. APPLICATION OF EDGE TRACKING AND MODELING: VISION-BASED POSE DETERMINATION 85
A. INTRODUCTION 85
B. PROBLEM STATEMENT 86
C. OVERVIEW 86
1. Sugihara's Formulation and Solution of Point Location Problem (PLP) 86
2. Camera Pose Determination as PLP 87
D. ALGORITHM 87
1. Finding Correct Position 90
2. Finding Correct Orientation 94
E. EXPERIMENTAL RESULTS 94
VIII. IMPLEMENTATION PLATFORM: YAMABICO-11 97
A. YAMABICO HARDWARE SYSTEM 97

B. ON-BOARD VISION SYSTEM 98

1. IMS Image Board 99

2. CCD Camera 100

C. MML SOFTWARE 100

D. IMAGE UNDERSTANDING SOFTWARE 101

1. Basic User-Level Image Functions 102

IX. CONCLUSIONS AND DIRECTIONS FOR FUTURE WORK 103

A. CONCLUSIONS 103

B. FUTURE WORK 104

APPENDIX A. ON-BOARD IMPLEMENTATION 105

APPENDIX B. WORLD MODEL 131

LIST OF REFERENCES 137

INITIAL DISTRIBUTION LIST 145

LIST OF TABLES

I. Approximation of detection probabilities for different n hits. 56
II. Data of line segments obtained by the tracking algorithm. 64
III. Number of line segments from several images. 65
IV. Number of vertical line segments from a sequence of three images. . . . 66
V. Yamabico' s main hardware specifications. 98
VI. Mapping of image modules to the VME bus address space. 100
VII. Summary of Cohu CCD camera capabilities. 100
VIII. Summary of basic user image functions. 102

LIST OF FIGURES

1. An image-understanding system. 3
2. An indoor environment image. 4
3. A gray-scale image. 13
4. Matrix representation of image plane. 14
5. A simple 6×6 gray-scale image. 14
6. Gray-level values of the 6×6 image. 14
7. A 16×16 window on an image of a polygonal object. 15
8. Gray-level values of the window pixels in the polygonal object image from Figure 7. 16
9. Gradient operators. 17
10. Indices of 3×3 pixel set centered at pixel $p = (x, y)$. 17
11. Gradient components as vectors. 18
12. A gradient image. 19
13. Gradient and edge directions. 19
14. Four distinct gradient regions in the gradient image. 20
15. Set of pixels and their fitted line 22
16. Two endpoints of a line segment L. 23
17. Principle of edge detection using gradient regions. 25
18. Main operations for edge detection. 28
19. Scanning direction. 29
20. General case of connectivity test. 30
21. Special cases of connectivity test. 30
22. A pixel is included in the region as one of its neighbors if it satisfies the connectivity test. 30
23. Current pixel starts a new region if it cannot belong to a region as one of its neighbors. 31

24. An image pixel data structure. 31
25. Current or previous row pixel data structure. 32
26. List structure of gradient regions. 32
27. Gradient region data structure. 33
28. Line segment data structure. 34
29. The main algorithm for edge detection by scanning. 35
30. Computing gradient magnitude with Sobel operator. 36
31. General connectivity test algorithm. 37
32. Least-squares fitting of a region. 37
33. Endpoints computation algorithm. 38
34. Input image of a printer. 39
35. Line segments extracted from a printer image. 39
36. Input image of a hallway portion. 40
37. Line segments detected form the hallway portion image. 40
38. Edge tracking in two opposite directions. 41
39. Edge pixel sequence with its fitted line segment. 42
40. Line segment detection by tracking. 43
41. Edge tracking and segment detection algorithm. 43
42. Finding tracking direction. 44
43. Tracking one side of the pixel sequence. 46
44. Next pixel set for each range $\mathcal{D}_i$. 47
45. Finding next pixel. 48
46. Consistency test. 48
47. Global algorithm for fast straight edge detection. 51
48. Distribution of first 10 hits using pseudo-random number generator with $c = 283 \times 646 + 181 = 182,999$. 54
49. Distribution of 3000 pixels using pseudo-random number generation with $c = 283 \times 646 + 181 = 182,999$. 55

50. An Image with total of M pixels having an m-pixels region. 56
51. An Image with total of M pixels having different regions. 57
52. Finding distance from point to line. 58
53. Pixel position in the L-coordinates 59
54. Algorithm for closeness test. 60
55. A square object test image. 60
56. Processed pixels before detecting all four line segments. 61
57. All four segments are detected after 608 hits. 61
58. Five line segments detected by 77 hits. 62
59. Ten line segments detected by 345 hits. 62
60. Fifteen line segments detected by 985 hits. 63
61. Twenty line segments detected by 2960 hits. 63
62. Number of random hits H_i to obtain first i segments. 65
63. Thirteen vertical edges detected by 2810 hits. 66
64. Twelve vertical edges detected by 2857 hits. 67
65. Fifteen vertical edges detected by 2553 hits. 67
66. Examples of polygons. 70
67. A polygon. 71
68. CW and CCW Polygon. 72
69. Data structure of a vertex in a polygon. 72
70. Data structure of a 4-vertex polygon. 72
71. A simple polygonal world. 73
72. Representation of 2D world data structure. 74
73. World coordinates and orientation of 2nd floor model. 75
74. 2D representation of the hallway. 75
75. A simple 3D world. 76
76. Representation of a door frame in the hallway model. 77
77. A simple $2\frac{1}{2}$D world. 78

78. Data structure of a vertex in a polygon within the $2\frac{1}{2}$D model. 79
79. Example of point visibility. 80
80. Truncated perspective viewing volume. 81
81. Steps of 3D world coordinates to 2D view projection. 81
82. Hallway view from x=2032 cm, y=124.25 cm, $\theta = 0°$, view angle=64° and focal length=3.0 cm. 82
83. Vertical edges of the hallway using $2\frac{1}{2}$D model from the same view point as in the previous figure. 83
84. Hallway view of the 3D model from x=3000 cm, y =124.25 cm, $\theta = 0°$. 83
85. Vertical edges of the hallway using $2\frac{1}{2}$D model from the same view point as in the previous figure. 84
86. Pose determination problem 86
87. The intersection of two circles passing through three control features uniquely determines the camera position. 88
88. Overview of a fast pose determination algorithm. 88
89. Vertical edges of model expectation 2D view. 89
90. Viewing angles from the camera position v to three positions on image plane lying on the three vertical image lines. 90
91. Viewing angles from v to three control features in the world. 91
92. Finding precise position of the camera. 91
93. Geometrical relation between the correct position v and the two circles passing through the three control features. 92
94. Geometry of finding camera orientation. 94
95. Algorithm of computing the correct camera orientation θ. 95
96. Estimated pose x=2040 cm, y= 120.0cm, $\theta = 0.0°$. 95
97. Correct pose x=2050.77 cm, y= 116.23, $\theta = 1.97°$. 96
98. Autonomous mobile robot, Yamabico-11. 97
99. Block diagram of Yamabico-11 hardware architecture. 99

100. User program description. 105
101. Structure of the readImage() function. 105
102. Edge detection program description. 106

ACKNOWLEDGMENTS

During the course of my research, there were many people who were instrumental in helping me. Without their guidance, help and patience, I would never have been able to accomplish my goal. These people deserve my appreciation.

First, I would like to thank my family, especially my parents, the first teachers in my life, for their guidance and encouragement, even thousands of miles from the United States. Their sincere prayers are always a spiritual force for me.

From my deepest heart, I would like to thank my wife Fadia, and our children children Ramy and Ranim, for their love, support and patience during many nights of work away at the office and lab.

I wish to express my deepest gratitude to Professor Yutaka Kanayama, whose help, support, guidance, and knowledge were driving factors for successful completion of this dissertation. He is always helpful to his students. He spends many hours in teaching and helping them to solve their research problems.

I am deeply indebted to my doctoral committee for their guidance, patience, and wisdom. Professor Craig Rasmussen generously provided me with needed mathematical guidance, as well as with proofreading the dissertation numerous times. The comments of Professor Thomas Wu, and advice of Professor Xiaoping Yun, were also exceptionally helpful. Many thanks to Professor Lynne Grewe for her help and support during critical periods of time in my research.

I appreciate the continuous help of the technical staff of the Computer Science Department, particularly of Mike Williams, Rosalie Johnson, Walter Landaker, and Valerie Brooks for their help in solving technical problems of the various computer systems.

I also wish to thank my fellow Ph.D. students from Egypt: Col. Nabil Khalil, LTC. Hesham Eldeeb, Maj. Hazem AbdElhamid, Maj. Ashraf Mamdouh, and Capt. Osama Kamal for their help and cooperation. Finally I wish to thank Dr. Mahmoud

Wahdan for his help during the last few years of his research at the Naval Postgraduate School.

I. INTRODUCTION

A. BACKGROUND

Computer vision is a field of science which deals mainly with the representation, extraction, and manipulation of objects and information from digital images. The goal is to devise a way for a computer to interpret images in a useful way. Naturally, what is useful will depend on the application at hand [Ref. 1]. A computer vision system is considered the enterprise of automating and integrating a wide range of processes and representations used for vision perception. It can be viewed in terms of the following main functional components [Ref. 2, 3]:

1. Image acquisition.
2. Image processing (transformation, encoding, compression, and transition of images).
3. Image understanding (image analysis, object detection, and pattern classification).
4. Geometrical modeling.
5. Output or display.

Image acquisition transforms the visual image of a physical object and its intrinsic characteristics into a set of digitized data which can be used by the processing unit of the system. This task can be considered as comprising illumination, image formation or focusing, sensing, and formatting camera output signal. *Image processing* is concerned with transforming an image in one storage area to a new (processed) image in another storage area. Some basic image-processing operations include feature enhancement, inversion, brightening, noise suppression, and compression. *Image understanding* mainly deals with the analysis of images for the purpose of extracting useful feature information. A key element in the task of locating features is that of **edge detection.** Identifying the locations of edges is essential for object recognition. *Geometric modeling*, a part of most of computer vision systems, describes the structures of objects related to the application domain for which the computer vision system was designed. *Output* capabilities are provided by most of the vision systems to assist the users in making process management decisions and verifying the results.

Over the past three decades, computer vision has evolved into many application domains. In each domain, a set of objects, tasks required, and knowledge sources should be specified. In the *aerial images domain*, examples of objects are terrain and buildings. The tasks required include resource analysis, weather prediction, spying, missile guidance, and tactical analysis. The knowledge sources for that domain are, for example, maps and geometrical models of shapes. In the *robotics domain*, the three-dimensional indoor or outdoor scenes and mechanical parts are examples of the objects related to this domain. Tasks required in such a domain include the identification of objects in the scene for obstacle avoidance, self-localization, and parts assembly. For the medical domain, the tasks are mainly diagnosis of abnormalities based on objects, such as body organs, and the design of anatomical models [Ref. 3].

B. IMAGE UNDERSTANDING

The ultimate goal in several computer vision applications is the extraction of useful information from the image data. The description, interpretation, and understanding of the features of an image scene altogether constitute *image understanding.* For example, a computer vision system with an image understanding capability used in industrial applications (such as assembly lines) should distinguish parts and list their features (as in size and number of holes). Moreover, the system can observe the parts, determine whether they are within specifications, and generate command signals according to the determined result. For a robot vision system, the system should help the robot recognize and interpret various objects and their spatial relationships in a scene. Motion control and execution should be performed through visual feedback from the vision system. Image understanding basically involves the study of *feature extraction, segmentation, classification*, and *interpretation.* A method used for both feature extraction and segmentation is *edge detection.* It is a fundamental problem in image understanding, and the simplest way to identify objects and obtain most of the relevant information about them such as shapes, locations, and distances from the camera. In this dissertation, we consider finding an efficient straight-edge detection method as a basis for autonomous vehicle visual navigation. Background information on edge detection will be presented in this chapter, and the principle of gradient-based edge detection will be presented in the next chapter, followed by the methods used. A typical image-understanding system is shown in Figure 1. The

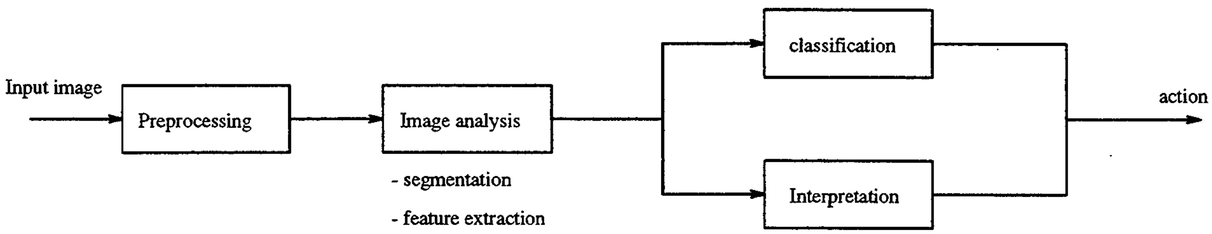


Figure 1. An image-understanding system.

input image is first preprocessed (for instance, by enhancement or proper representation of the data) [Ref. 4]. The image is then analyzed using segmentation and feature extraction. The results are fed into a classifier or an interpreter in order to obtain useful information, such as the relationship between different objects in the scene, and to let the system take the proper action.

1. Feature Extraction

In the process of feature extraction, it is necessary to extract certain features of objects from the scene. In an indoor environment image, as shown in Figure 2 for instance, the system should be able to extract specific features of the indoor structures, such as the door locations in the hallway. These features serve as *landmarks* for visual navigation tasks. By way of comparison, in an X-ray image, the gray-level amplitude represents the absorption characteristics of the body masses which enables the discrimination of bones from tissues, or of healthy tissues from diseased tissues [Ref. 4].

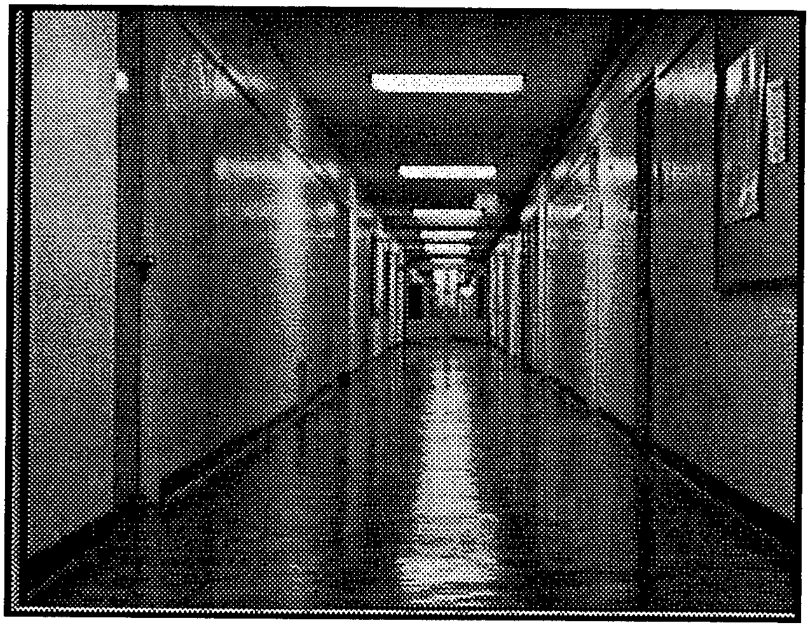


Figure 2. An indoor environment image.

2. Segmentation

The segmentation technique is required by a vision system in order to isolate the objects from the background. Segmentation is often imagined as the splitting of the image into a number of regions, each having a high level of uniformity in some designated parameter such as brightness, color, or texture [Ref. 5]. One method of segmentation is the *region-growing* technique [Ref. 6]. The goal of region-growing is to use image characteristics to map individual pixels in an input image to sets of pixels, called *regions* [Ref. 3]. In this technique, pixels are placed in a region on the basis of their similar intensity. Similar adjacent regions are then merged sequentially to form larger regions [Ref. 4]. This technique is found to be useful with the aid of edge detection. However, the technique tends to be quite computationally intensive [Ref. 5]. Another method of segmentation is that of *template matching* where an image is matched against templates from a given list in order to locate objects [Ref. 4]. Another method of segmentation is *thresholding* the image at a particular intensity level. This method results in setting the objects as black figures on a white background according to binary images [Ref. 5].

3. Classification and Interpretation

These tasks of image understanding are concerned with the ability of the vision system to interpret the obtained information from segmentation or feature extraction. This is done to provide a description of the objects in an image scene in a useful way. Several techniques are used in these tasks, including pattern recognition, supervised classification, statistical classification, clustering, decision trees, and similarity measures [Ref. 4]. An important method of interpretation by an image understanding system for autonomous mobile robots is that of matching between line segments (edges) of objects in an image and those of an indoor structure.

C. AUTONOMOUS VEHICLES AND VISION

Among the real-world applications of Artificial Intelligence are those of the autonomous mobile robot vehicles. They are defined as those vehicles that are capable of intelligent motion and action without requiring further human intervention [Ref. 8]. Many individual components and research fields can be integrated for autonomous vehicle development. Such components include motion planning (for example in [Ref. 9, 10, 11, 12]), image understanding (as in [Ref. 13, 14]), control (some examples are in [Ref. 15, 16]), kinematics [Ref. 17], sonar technology [Ref. 18, 19] , electronic circuits, system architecture, programming, neural networks [Ref. 20], and fuzzy logic and control [Ref. 21].

Research in the area of autonomous vehicles is of major practical interest. There are several potential applications of autonomous vehicles, such as manufacturing, construction, waste management, space and undersea exploration, assistance for the disabled, intelligent wheel-chairs for the handicapped, medical surgery, remote repair and maintenance, military operations, and material-handling system for offices and factories [Ref. 8, 22].

In military applications, emphasis is placed on using autonomous vehicles for handling a myriad of hazardous duty assignments [Ref. 9] like mine searching, UXO clearing, reconnaissance, and fire fighting. A semi-autonomous vehicle called *Shepherd* is under development at the Naval Postgraduate School for research in mine detection and UXO clearing.

Many of the above tasks require image understanding as an essential component, especially for visual navigation and guidance control of the vehicles in either outdoor or indoor environments. We define the visual navigation of an autonomous vehicle as "the ability of an autonomous vehicle to perform efficiently its global and local motion planning in a partially known environment under the guidance of a vision sensor". The visual navigation of autonomous vehicles in an indoor environment involves several aspects. Each can be considered as a stand-alone problem. Image analysis, model interpretation, pose estimation, object recognition, and obstacle avoidance are examples of these problems.

1. Vision-Guided Navigation

Vision-guided navigation was the focus of a number of researchers. Lebegue and Aggarwal [Ref. 23] developed an algorithm for automatically constructing CAD models of a structured scene as imaged by a single camera on a mobile robot. The scene to be modeled is assumed to be composed mostly of linear edges with particular orientations in 3D. Bishay, Peters and Kawamura [Ref. 14] developed a system that can detect stationary objects in an indoor scene, using log-polar mapping for the purpose of building a map for the environment to guide the robot from one place to another. This system was developed so that the size of the log-polar map is 64×64 pixels. This is not suitable for the high-resolution images currently being used. Buger and Bhamus [Ref. 24] investigated the problem of estimating the camera motion parameters from a 2D image sequence that has been obtained under combined 3D camera translation and rotation. They solved the problem using the concept of "Focus Of Expansion (FOE) feasibility". Taylor and Kriegman presented in [Ref. 25] an algorithm that would enable a mobile robot equipped with a visual recognition system to carry out a systematic exploration of an unfamiliar environment in search of one or more recognizable targets. Huttenlocher, Leventon, and Rucklidge presented in [Ref. 26] a method of using a sequence of monocular images for navigating a robot from an initial position to a specified landmark in its visual field. Lazanas and Latombe [Ref. 27] investigated the problem of developing navigation algorithms that rely on the ability of the robot to recognize and localize a specific set of landmarks at known positions in the environment. The problem of exploring an unknown polygonal room with a bounded number of polygonal obstacles was investigated in [Ref. 28] by Deng et al. Other applications of image understanding as an essential component have been recognized in many areas. For example, a mobile robot system is presented in [Ref. 29], on which a human can ride and share access to the environment through visual navigation. In [Ref. 30] a robotic system for guiding the blind on the road or sidewalk is presented.

Some projects were dedicated for the research in mobile robot vision. The FINALE vision-guided mobile robot system, which navigates in an indoor environment at a speed of 10 meters per minute in the presence of obstacles, was presented in [Ref. 13, 31]. This model-based system matched landmarks in the scene with features extracted from the images, to perform self-localization. However, it was limited to stop-and-start motion, since the robot had to be motionless to obtain an accurate video image. The NAVLAB [Ref. 7] is a commercial truck converted into a robot vehicle. It contains several on-board general-purpose computer workstations, as well as a WARP parallel architecture computer. Its sensors include color TV cameras and an ERIM laser range finder. Another approach to visual modeling and recognition is concerned with developing techniques for automatically building a representation of a complex physical environment (e.g., natural terrain populated with objects such as roads, bridges, bushes, and trees) based on data from imaging sensors and previously stored knowledge. UMass Mobile Perception Laboratory (MPL) [Ref. 32] is a similar effort to build a landmark-based autonomous vehicle system composed of several independent processes, each solving a particular aspect of the navigation problem.

These are then integrated into a fully capable autonomous vehicle for both on-road and cross-country navigation. The integrated system was designed to "react" in real time to a changing environment and to "reason" about ways to achieve its goals. This autonomous vehicle performs pose determination in 30 seconds to 1 minute, which is very slow in real time. At 5 MPH an obstacle at 30 feet can be avoided if the obstacle is at least 2 feet high.

The major problem with image understanding tasks in robotics is the expensive computation time required for image processing and then for image analysis. The computation time is very critical, since slow performance of image understanding algorithms leads to hazardous situations in real time. Developing efficient image understanding methods is one of the more challenging problems in both computer vision and robotics research.

D. PROBLEM STATEMENTS

The objective in this dissertation is to investigate a new efficient model-based image understanding method for an indoor autonomous vehicle, for both military and industrial applications, through a fast straight-edge detection algorithm.

This work is part of the research project for the Naval Postgraduate School's autonomous mobile robot vehicle **Yamabico-11**. Several previous efforts in motion planning [Ref. 9, 10, 11], automated cartography [Ref. 33], sonar-based obstacle avoidance [Ref. 34], and image understanding [Ref. 35, 36, 37, 38] have made contributions to the project. We expect this research to produce significant and efficient methods to integrate image understanding with previous efforts and the current software system on the robot.

1. Problem Definition

The problem addressed herein is: How can the model-based image understanding task for an autonomous vehicle be accelerated? The problem is decomposed into the following sub-problems:

1. How can the straight-edge detection, as a basic function in image understanding, be performed efficiently?
2. How can the vehicle's world environment be modeled so that its interpretation function within the image-understanding task is executed rapidly?
3. How can the correct pose of the vehicle be found efficiently, given the constructed model, the estimated pose, and the camera parameters?

2. Assumptions

The following assumptions and limitations were incorporated into the research in order to focus the work on the image-understanding problem:

- A single CCD (Charge-Coupled Device) vision system is used.
- The camera parameters (image size, focal length, physical size of CCD image plane) and field of view are already set and known.
- The vehicle is operating in a partially known, orthogonal indoor world: walls, ceilings, door frames and floor meet at right angles. (These are perfectly suitable for the current operating environment of the testbed vehicle Yamabico-11, which is the second floor of Spanagel Hall at the Naval Postgraduate School.)
- The vehicle has dead reckoning capability.
- The vehicle's operating environment is bounded by a clockwise polygon (see Chapter VI).

E. PREVIOUS WORK

1. Edge Detection

Many researchers have worked toward solving the straight edge detection problem. The ability to solve this problem is required in robotics and in other areas. For instance, matching between straight edges in an image plane and those in a world model is especially important in model-based real-time robot navigation and in object recognition.

The approaches to the problem of edge detection vary from one application to another. Some are highly dependent on the knowledge of the images being processed.

Bolles [Ref. 39] described the idea of adjusting a boundary estimate by carrying local searches out at regular intervals along directions perpendicular to the estimated boundary. Another method was developed for some medical applications [Ref. 40] using divide-and-conquer for boundary detection. The Hough transformation for detecting lines and curves, introduced in [Ref. 41], is considered one of the standard techniques. Although this method by itself does not give the endpoints of the detected edges, Dudani and Luck [Ref. 42] included a least-squares fitting procedure into this method to obtain the endpoints of the detected line. Martelli proposed a heuristic search method to follow edges in images and, by doing so, converted the problem of edge finding into a graph searching problem [Ref. 43] using the A^* algorithm [Ref. 44, 45]. Some applications of dynamic programming were described in [Ref. 3, 46]. The contour-following method can find regions for a given binary image even without any knowledge on the boundary shape [Ref. 47]. The method was adapted to gray-level images, as described in [Ref. 48]. The algorithm presented in [Ref. 49] requires scanning the whole image to obtain line-support regions and then determine the location and properties of the edges. Some other researchers followed this approach with slight modifications [Ref. 50, 51]. Other methods also used the whole image scanning strategy with different edge detectors and modules [Ref. 52, 53]. A contour-tracing algorithm was proposed in [Ref. 54], based on a priori knowledge about the edges to be searched. Marr and Hildreth [Ref. 55] discussed a theory of edge detection using a two-dimensional Gaussian operator. Although this theory was highly influential in the following few years, it has the problem of expensive computational time since it requires preprocessing the images in order to study them at different scales [Ref. 5]. The computational approach to edge detection by J. Canny [Ref. 56] has emerged from this approach with emphasis on the optimality of the detector at any scale and on dealing with different signal-to-noise ratios in the image. He formulated the criteria desired in any edge detection operator as good detection, good localization, and only one response to a single edge. A variation of the Canny operator is presented in [Ref. 57] to optimize the composite criteria using the calculus of variations. These approaches, and others discussed in [Ref. 5], are concerned with signal-to-noise ratios and the accuracy with which edge magnitude and orientation can be estimated. Davies and Johnstone [Ref. 58, 59] used the Sobel operator in the edge-detection task on enhanced images of their project, with slow execution time. A general discussion about using parallel processing and hardware implementation to speed the image processing tasks is found in [Ref. 5]. However, our approach in speeding up the edge detection is based on a single processor computer system.

2. Pose Determination

For an autonomous vehicle moving in a certain environment, some positional errors occur and accumulate. Thus, this problem of robot localization (pose determination) has received the attention of several researchers in the mobile robot field. Some used model-sonar based navigation [Ref. 11, 60] to solve the problem. We will give a brief summary of previous efforts in the field of visual navigation.

Fischler and Bolles [Ref. 61] solved for the lengths of rays from the optical center of the camera to the points in 3D space. The closed-form solution they presented is quite complex. Using triangle pairs and the Hough transform, Linnainmaa et al. solved the problem of finding the coordinates of the 3D points in camera coordinates [Ref. 62]. The approach used in [Ref. 63] was to decompose the solution into two stages, for rotation first and then for translation, with another method for simultaneous rotation and translation. With the scanning method for edge detection and a complete 3D model, Peterson [Ref. 35] used Sugihara's principle [Ref. 64] in the pose determination. Using a CCD video camera coupled with a conic reflector, Pegrad and Mouaddib [Ref. 65] obtained omni-directional views and presented a method for localization from natural landmarks obtained in the navigation area. Another approach was taken in [Ref. 66] by converting a sequence of image measurements into a representation of the robot's pose. Although this method does not require explicit environment modeling, it has the overhead of a collection of training examples, each of which specifies the video image observed when the robot is at a particular location and orientation.

F. ORGANIZATION OF DISSERTATION

The remainder of this dissertation is organized as follows: Chapter II discusses the principles of image representation and edge detection as a basic function for several image understanding algorithms. Chapter III presents edge detection using the concept of *gradient regions* by image scanning. In Chapter IV, we present a novel and robust algorithm for edge detection using *direction-controlled edge tracking.* Chapter V describes a global method for detecting all important edges by introducing the concept of *random hitting.* In this method, finding an edge multiple times is eliminated. Chapter VI describes world modeling of the operating environment of an autonomous vehicle, and shows a simple method for modeling the environment by what is called a $2\frac{1}{2}$D model. Chapter VII discusses how to use this model and the robust edge tracking method to perform pose determination. In Chapter VIII, we give a brief description of the hardware/software system of the autonomous mobile robot vehicle Yamabico-11 at the Naval Postgraduate School, with emphasis on the image system mounted on the robot. Chapter IX summarizes the contributions of this dissertation with recommendations for future work.

II. PRINCIPLE OF STRAIGHT-EDGE DETECTION USING GRADIENTS

A. IMAGE REPRESENTATION

An *image function* is a mathematical representation of an image. Generally, a gray-scale image $\mathcal{I}$ is represented in terms of the intensity (or brightness of the gray level) $f(p)$ for each pixel $p = (x, y)$. Such a gray-scale image is W pixels wide and H pixels high. An example of a gray-scale image is shown in Figure 3. Thus, an image

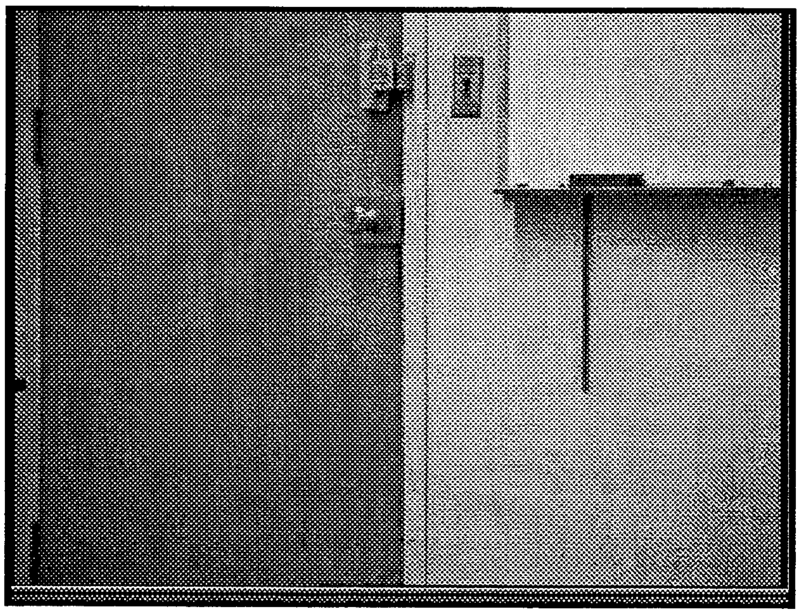


Figure 3. A gray-scale image.

is described by means of a $W \times H$ matrix of nonnegative integer values $f(x, y)$ that indicate the light intensity of the pixel with coordinates (x, y), as shown in Figure 4, where the origin (position (0,0)) of the matrix is at the left bottom corner. This makes the matrix correspond to the pixel coordinates in the image plane, although this may not be compatible with the known definition of matrices. The range of values assigned to the gray level of pixels in images usually depends on the number of bits to be used for representing the gray-scale values. Thus, the total number of levels in a gray-scale image is usually a power of 2. Most computer systems now

$$\begin{bmatrix} f(0,H-1) & f(1,H-1) & \ldots & f(W-1,H-1) \\ . & . & \ldots & . \\ . & . & \ldots & . \\ . & . & \ldots & . \\ f(0,1) & f(1,1) & \ldots & f(W-1,1) \\ f(0,0) & f(1,0) & \ldots & f(W-1,0) \end{bmatrix}$$

Figure 4. Matrix representation of image plane.

support representation of eight bits for the gray-scale values. Thus, they range from 0 to $2^8 - 1$ (or 255). The value 0 is assigned for black, and 255 for white. For the 6×6 image shown in Figure 5, the corresponding gray-scale pixel values are shown in the image matrix in Figure 6.

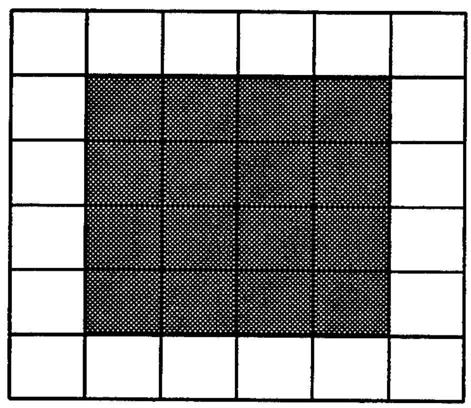


Figure 5. A simple 6×6 gray-scale image.

$$\begin{bmatrix} 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 20 & 20 & 20 & 20 & 255 \\ 255 & 20 & 20 & 20 & 20 & 255 \\ 255 & 20 & 20 & 20 & 20 & 255 \\ 255 & 20 & 20 & 20 & 20 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

Figure 6. Gray-level values of the 6×6 image.

For a 16×16 pixel window on an image of a polygonal object, as shown in Figure 7, the actual gray-level values of its pixels are shown in Figure 8.

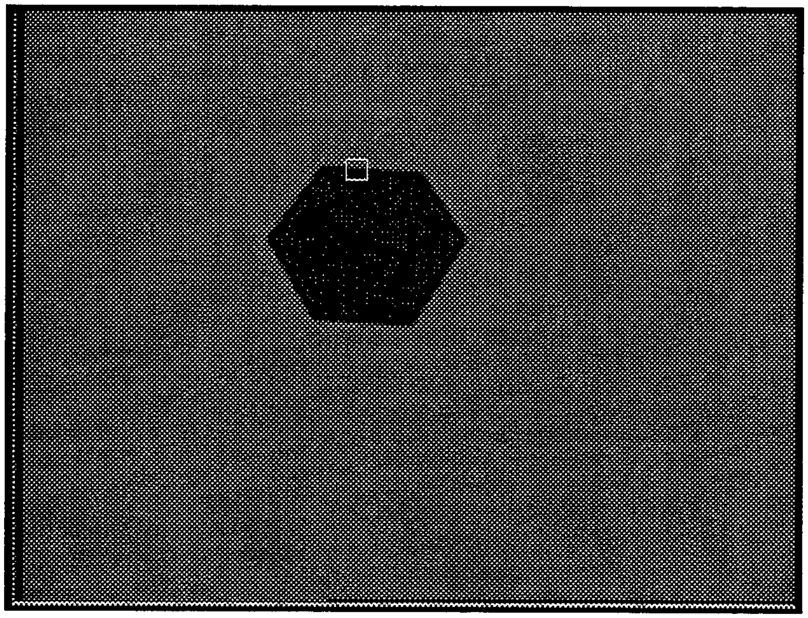


Figure 7. A 16×16 window on an image of a polygonal object.

B. EDGE DETECTION PROBLEM

The problem of fundamental importance in image analysis is the edge detection problem. Usually, edges in an image are boundaries of objects. Thus, edge detection is the simplest way to identify important contours and objects in the image. Edges are curves or lines in the image plane across which there is an abrupt change in brightness. By using the difference in gray level between image pixels, we can locate the boundaries of objects. By doing so, most of the relevant information about object shape, location, and distance from camera can be obtained. For instance, the image in Figure 5 has four edges.

A *gradient region* is a small region in an image where light intensities are changing rapidly. It is defined as an intensity discontinuity in the image [Ref. 3]. The problem now is how to locate those regions in a given image, and then find

$$
\begin{bmatrix}
95 & 94 & 95 & 95 & 95 & 92 & 90 & 93 & 95 & 97 & 94 & 93 & 95 & 99 & 98 & 98 \\
96 & 95 & 96 & 99 & 98 & 92 & 87 & 92 & 94 & 92 & 94 & 98 & 99 & 100 & 98 & 98 \\
93 & 94 & 91 & 90 & 94 & 96 & 95 & 96 & 100 & 99 & 94 & 95 & 96 & 98 & 98 & 100 \\
93 & 92 & 93 & 94 & 95 & 95 & 92 & 87 & 89 & 93 & 90 & 92 & 97 & 94 & 93 & 95 \\
98 & 100 & 92 & 85 & 89 & 92 & 94 & 94 & 94 & 95 & 95 & 96 & 99 & 98 & 92 & 94 \\
95 & 94 & 95 & 96 & 95 & 94 & 96 & 92 & 88 & 89 & 91 & 95 & 98 & 98 & 97 & 95 \\
96 & 100 & 97 & 94 & 94 & 94 & 95 & 93 & 91 & 94 & 95 & 96 & 98 & 98 & 91 & 91 \\
99 & 97 & 98 & 99 & 98 & 98 & 99 & 96 & 92 & 92 & 96 & 99 & 100 & 102 & 99 & 94 \\
95 & 101 & 103 & 95 & 90 & 94 & 96 & 97 & 99 & 100 & 97 & 98 & 101 & 96 & 93 & 94 \\
94 & 90 & 89 & 96 & 99 & 98 & 98 & 99 & 99 & 97 & 96 & 95 & 98 & 101 & 103 & 100 \\
66 & 76 & 79 & 71 & 70 & 72 & 73 & 75 & 79 & 77 & 78 & 83 & 88 & 86 & 87 & 91 \\
41 & 34 & 38 & 51 & 50 & 47 & 48 & 51 & 55 & 58 & 56 & 56 & 61 & 65 & 63 & 644 \\
15 & 19 & 22 & 21 & 20 & 20 & 23 & 24 & 27 & 30 & 30 & 31 & 34 & 36 & 38 & 377 \\
7 & 0 & 2 & 11 & 11 & 8 & 6 & 7 & 13 & 15 & 10 & 9 & 16 & 17 & 11 & 11 \\
13 & 12 & 10 & 10 & 10 & 10 & 9 & 9 & 7 & 7 & 6 & 6 & 7 & 9 & 8 & 7 \\
26 & 23 & 22 & 22 & 23 & 21 & 17 & 16 & 18 & 16 & 12 & 14 & 20 & 19 & 13 & 10
\end{bmatrix}
$$

Figure 8. Gray-level values of the window pixels in the polygonal object image from Figure 7.

their corresponding edges. In the following two sections, a solution to this problem is presented.

C. GRADIENT DIRECTIONS AND GRADIENT REGIONS

1. Gradient Computation

A basic and common step in the process of detecting edges in a gray-scale image is to compute the gradient at each pixel. The partial derivatives $\frac{\partial f(p)}{\partial x}$ and $\frac{\partial f(p)}{\partial y}$ of the intensity function f of a pixel p with respect to x and y can be computed using a *gradient operator*. A gradient operator is represented by a pair of masks, Δ_x and Δ_y. Each mask is a square matrix of weights mapped onto a group of pixels around an origin (or center) pixel. The mask Δ_x will be used for computing the horizontal direction gradient, and Δ_y for the vertical direction gradient. There are several edge operators that can be used in edge detection [Ref. 67]. Figure 9 shows three of the well-known operators. In Figure 10, we show 3×3 pixel locations. The eight

Operator	Δ_x	Δ_y
Roberts	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Prewitt	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Figure 9. Gradient operators.

surrounding pixels about a center pixel at location $p = (x, y)$ will be used to find the gradient information for pixel p.

$$\begin{bmatrix} (x-1, y+1) & (x, y+1) & (x+1, y+1) \\ (x-1, y) & (x, y) & (x+1, y) \\ (x-1, y-1) & (x, y-1) & (x+1, y-1) \end{bmatrix}$$

Figure 10. Indices of 3×3 pixel set centered at pixel $p = (x, y)$.

The two gradient components $g_x(p) = \frac{\partial f(p)}{\partial x}$ and $g_y(p) = \frac{\partial f(p)}{\partial y}$ can be computed using the Sobel operator by multiplying each of the weights corresponding to those pixels in S on each mask with its intensity value.

$$\begin{aligned} g_x(p) &= -f(x-1, y-1) - 2f(x-1, y) - f(x-1, y+1) \\ &\quad +f(x+1, y-1) + 2f(x+1, y) + f(x+1, y+1), \\ g_y(p) &= f(x-1, y+1) + 2f(x, y+1) + f(x+1, y+1) \\ &\quad -f(x-1, y-1) - 2f(x, y-1) - f(x+1, y-1). \end{aligned}$$

Then the gradient magnitude $g(p)$ is computed from these two orthogonal gradient components (Figure 11) as follows:

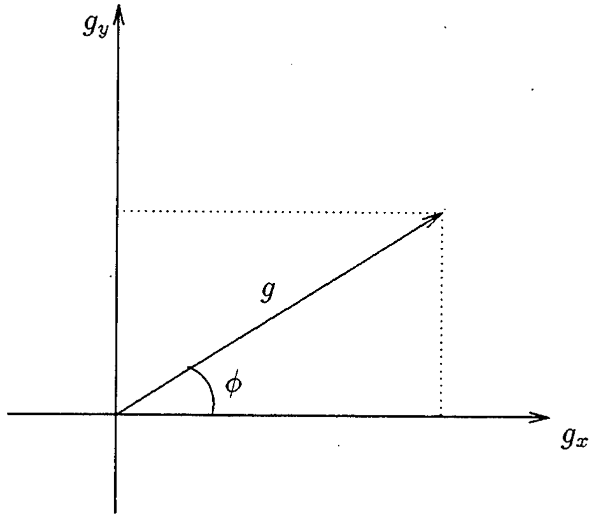


Figure 11. Gradient components as vectors.

$$g(p) = \sqrt{\left(g_x(p)\right)^2 + \left(g_y(p)\right)^2},$$

when $g(p) > 0$, its *gradient direction* $\phi(p)$ is defined as

$$\phi(p) = atan2\Big(g_y(p), g_x(p)\Big).$$

The function $atan2\Big(g_y(p), g_x(p)\Big)$ gives the direction in four quadrants $[-\pi, \pi]$. It is more useful than the mathematical equation $tan^{-1}\Big(g_y(p)/g_x(p)\Big)$ because the latter has range $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and is not defined when $g_x(p) = 0$ [Ref. 68]. The gradient direction information of pixels is extremely valuable in the straight edge detection task, as shown later.

A pixel p is said to be *significant* if its gradient magnitude $g(p)$ is greater than some threshold value. The gradient image (the image with only pixels of significant gradient magnitudes) obtained from the original gray-scale image in Figure 3 is shown in Figure 12.

2. Gradient Regions

Let ψ be the direction of a straight edge L in an image, and let p be a pixel in L (Figure 13). Then the gradient direction $\phi(p)$ must be approximately orthogonal to ψ. The relation between the edge direction ψ and the gradient direction $\phi(p)$ is

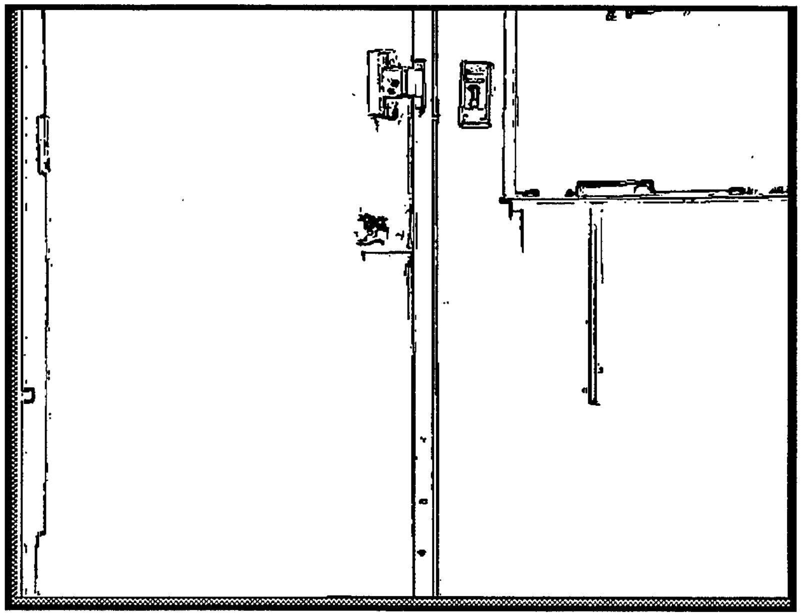


Figure 12. A gradient image.

defined by $\phi(p) \approx \psi + \frac{\pi}{2}$ or $\psi - \frac{\pi}{2}$. Therefore, all pixels in one straight edge must have a similar gradient direction ϕ. We say "similar" because noise can affect the value of $\phi(p)$. Through this observation, we can define a "gradient region". A *gradient region* R in an image $\mathcal{I}$ is a set of pixels that satisfies the following conditions:

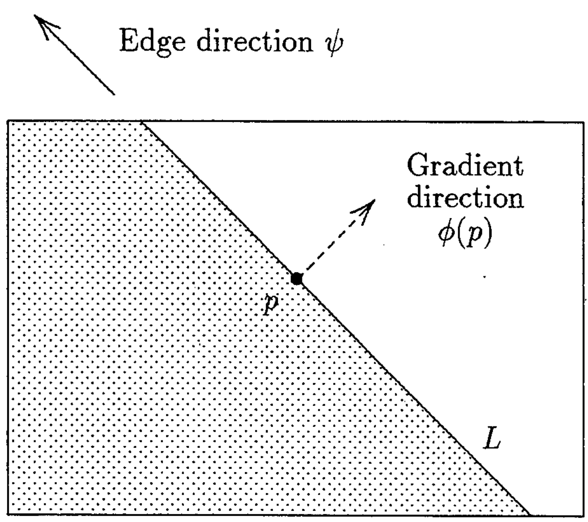


Figure 13. Gradient and edge directions.

(1) Each pixel p in R has a significant gradient magnitude $g(p)$.

(2) Each pixel p in R has a similar gradient direction $\phi(p)$.

(3) All pixels in R are connected by eight-neighborhood (any two pixels in R must be adjacent within a 3×3 window and one of them is the center of this window).

Figure 14 is a blowup of a part of Figure 12, including boundaries of the light switch beside the door lock. It shows four distinct gradient regions, and each pixel in

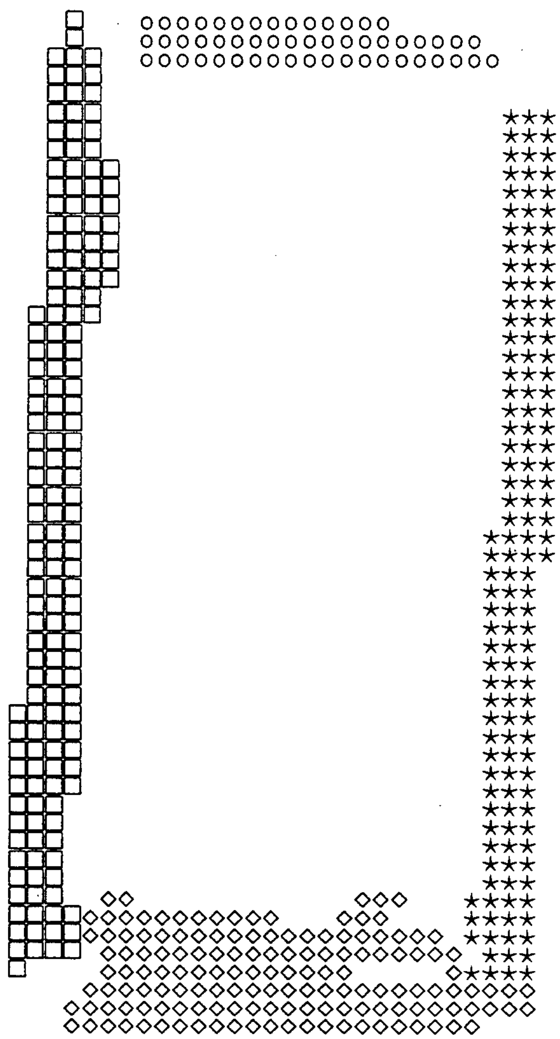


Figure 14. Four distinct gradient regions in the gradient image.

a particular gradient region R is represented by a unique symbol.

D. FINDING EDGE FEATURES BY LEAST-SQUARES FITTING

Assuming that we find an edge region, the next task is to find its linear features. The least-squares linear fitting algorithm [Ref. 69] is a very robust and helpful method for doing this. Although it was originally used to find line segments from sonar return data for mobile robot navigation, it can be used in image understanding as well. For a set R of pixels representing one gradient region, where $R = \{p_1, ..., p_n\}$ and $p_i = (x_i, y_i)$, we obtain the moments m_{jk} by

$$m_{jk} = \sum_{i=1}^{n} {x_i}^j {y_i}^k,$$

where $0 \leq j, k \leq 2$ and $j + k \leq 2$. Actually, $m_{00} = n$.

The secondary moments M_{ij} around the centroid $C = (\mu_x, \mu_y)$ where $\mu_x = \frac{m_{10}}{m_{00}}$ and $\mu_y = \frac{m_{01}}{m_{00}}$ are

$$M_{20} \equiv \sum_{i=1}^{n} (x_i - \mu_x)^2 = m_{20} - \frac{m_{10}^2}{m_{00}},$$

$$M_{11} \equiv \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y) = m_{11} - \frac{m_{10} m_{01}}{m_{00}},$$

$$M_{02} \equiv \sum_{i=1}^{n} (y_i - \mu_y)^2 = m_{02} - \frac{m_{01}^2}{m_{00}}.$$

A pixel $p = (x, y)$ that satisfies

$$x \cos \alpha + y \sin \alpha = r \tag{II.1}$$

lies on a line L whose normal has an orientation α and whose distance from the origin is r (Figure 15). We represent this line L by (α, r). This representation has a striking advantage as opposed to the usual method of using a formula $y = f(x)$, because the former method has no difficulty in expressing lines that are perpendicular to the x axis. The signed distance (or residual) δ_i from a pixel $p_i = (x_i, \ y_i)$ to the line $L = (r, \alpha)$ is

$$\delta_i = x_i \cos \alpha + y_i \sin \alpha - r.$$

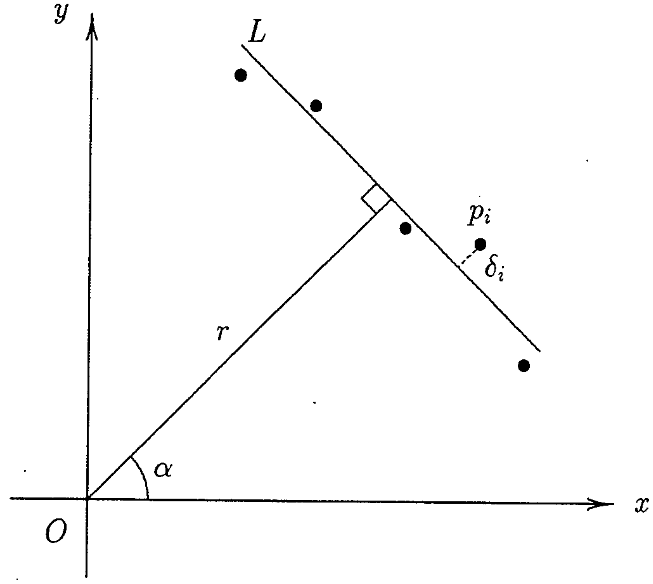


Figure 15. Set of pixels and their fitted line

Therefore, the sum of the squares of all the residuals is

$$S = \sum_{i=1}^{n}\Big((x_i \cos\alpha + y_i \sin\alpha) - r\Big)^2.$$

Since the line which best fits the set of points is supposed to minimize S, the optimum line (r, α) must satisfy

$$\frac{\partial S}{\partial r} = \frac{\partial S}{\partial \alpha} = 0.$$

Thus,

$$\begin{aligned}
\frac{\partial S}{\partial r} &= -2\sum_{i=1}^{n}\Big((x_i \cos\alpha + y_i \sin\alpha) - r\Big) \\
&= 2\left(r\sum_{i=1}^{n} 1 - \cos\alpha \sum_{i=1}^{n} - \sin\alpha \sum_{i=1}^{n} y_i\right) \\
&= 2(r\ m_{00} - m_{10}\cos\alpha - m_{01}\sin\alpha) = 0,
\end{aligned}$$

and

$$r = \frac{m_{10}}{m_{00}}\cos\alpha + \frac{m_{01}}{m_{00}}\sin\alpha = \mu_x \cos\alpha + \mu_y \sin\alpha, \qquad \text{(II.2)}$$

where r may be negative. Substituting r in Eq. II.1 by Eq. II.2, we obtain

$$\frac{\partial S}{\partial \alpha} = 2\sum_{i=1}^{n}\Big((x_i - \mu_x)\cos\alpha + (y_i - \mu_y)\sin\alpha\Big)\Big(-(x_i - \mu_x)\sin\alpha + (y_i - \mu_y)\cos\alpha\Big)$$

$$= 2\sum_{i=1}^{n}\left((y_i-\mu_y)^2-(x_i-\mu_x)^2\right)\sin\alpha\cos\alpha$$

$$+2\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)(\cos^2\alpha-\sin^2\alpha)$$

$$= (M_{02}-M_{20})\sin 2\alpha + 2M_{11}\cos 2\alpha = 0.$$

Therefore,

$$2\alpha = \operatorname{atan2}(-2M_{11},\ M_{02}-M_{20}). \tag{II.3}$$

Note that, by Eq. II.3, the value of 2α can be in any quadrant, and so $\alpha \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Eqs. II.2 and II.3 are the solutions to the least-squares fitting problem.

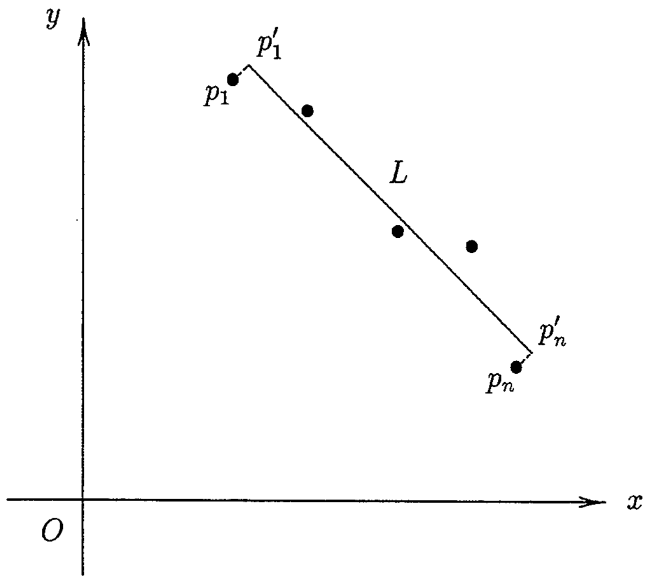


Figure 16. Two endpoints of a line segment L.

Since the residual δ_i of a pixel $p_i = (x_i, y_i)$ is given by

$$\delta_i = x_i\cos\alpha + y_i\sin\alpha - r,$$

the projection, p_i' of the point p_i onto the fitted line is

$$p_i' = (x_i - \delta_i\cos\alpha,\ y_i - \delta_i\sin\alpha). \tag{II.4}$$

We will use p_1' and p_n' as estimates of the endpoints of the line segment L obtained from the set of data points R (Figure 16) [Ref. 68].

E. SUMMARY

The principle of edge detection using gradient regions is described in this chapter. The principle is illustrated in Figure 17. The first image, the original one, has three areas with different intensities. There are three gradient regions separating them. The line segments corresponding to these gradient regions are then obtained using the least-squares fitting, as described in the previous section. This principle can be applied in several ways. In the next chapter we discuss the image-scanning method using this principle.

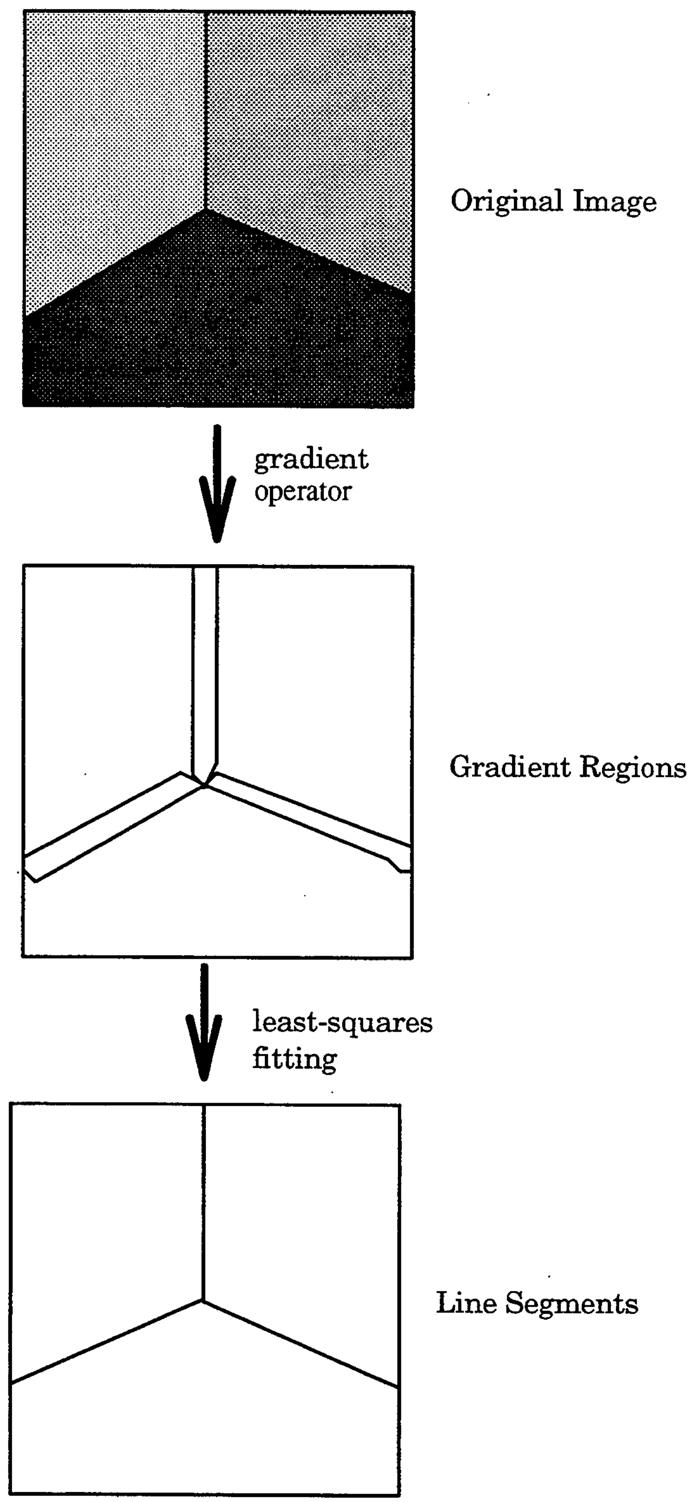


Figure 17. Principle of edge detection using gradient regions.

III. STRAIGHT-EDGE-FINDING METHOD BY SCANNING

The principle of edge detection described in the previous chapter can be applied in different ways. In this chapter, one such method is presented. In this method, the whole image is scanned pixel-by-pixel to find those pixels that form gradient regions and hence the line segments that represent those gradient regions. Figure 18 shows the conceptual view of the operations performed to detect edges using this method. This method is different from that used in [Ref. 35] in the technique for forming gradient regions. The method includes a simpler approach and different data structures for that purpose, as we will describe. The process is divided into two levels:

- Pixel-level operation: to decide whether each pixel is an "edge" pixel or not, and to decide in which region this pixel should be included.
- Region-level operation: to find the line segment information for each gradient region.

In this chapter, an overview of the method is presented, followed by a description of the algorithm and data structures. Finally, a sample of the experimental results is presented.

A. IMAGE SCANNING

The image is scanned starting from the pixel at (1,1), by excluding the pixels that exist at the image outer boundaries. For an image with size of $W \times H$ pixels, the pixels on rows 0 and $W-1$ are excluded, as well as those on columns 0 and $H-1$, because the Sobel edge operator we are using (see chapter II) requires eight surrounding pixels about the central pixel. The scanning is performed from left to right and from bottom to top, as shown in Figure 19.

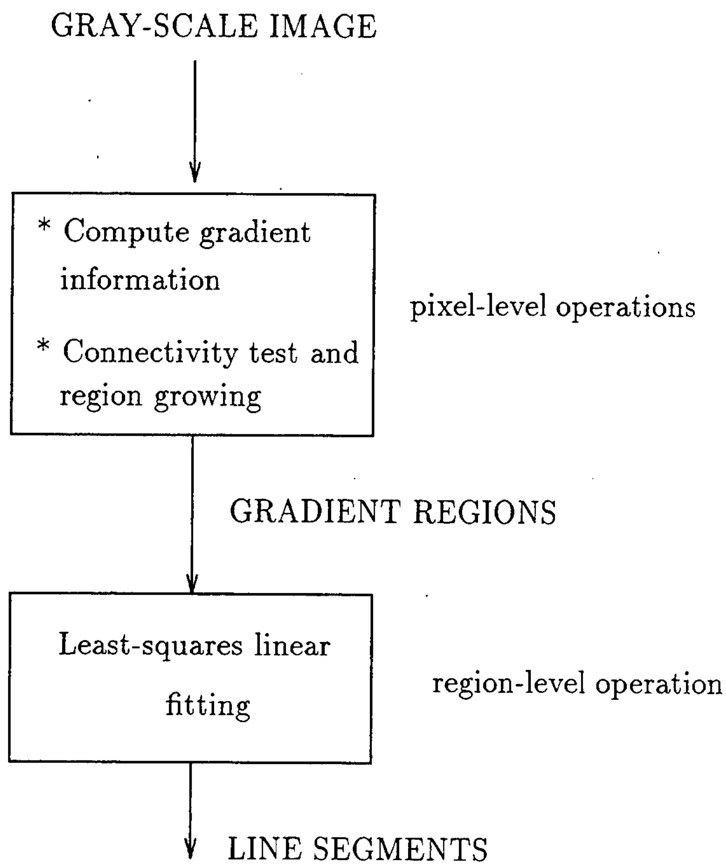


Figure 18. Main operations for edge detection.

B. CONNECTIVITY TEST

During the scanning process, it is necessary to test whether the pixel has significant gradient magnitude $g(p)$ or not. If so, its gradient direction $\phi(p)$ will be computed. This pixel then becomes a candidate to be a part of some gradient region in the image. Finding this region for a specified pixel is the subject of this section. One characteristic of an edge region is the connectivity. The *connectivity test* of a pixel is intended to see if the pixel can be connected to some gradient region containing one of its neighbors. If the new scanned pixel $p = (i, j)$, such that $1 < i < W - 2$ and $1 < j < H - 1$, then the connectivity test is done for the pixels at $(i-1, j)$, $(i-1, j-1)$, $(i, j-1)$, and $(i+1, j-1)$, which means that the test includes the pixel immediately to the left of the current pixel and the three nearest pixels in the

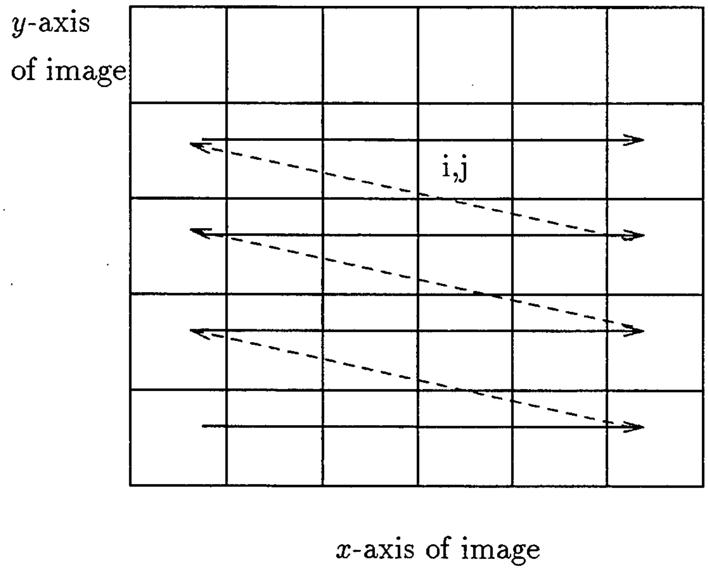


Figure 19. Scanning direction.

next lower row. This situation is shown in Figure 20, where the current pixel at (i, j) is represented by a bullet and the pixels indicated by shaded squares are those considered in the connectivity test. However, there are some special cases that should be considered for the connectivity test. For pixels at locations with $j = 1$, the test will be done with only one pixel to the left. For pixels at locations with $i = 1$, the test will be done only with the two pixels $(i, j - 1)$ and $(i + 1, j - 1)$. For the pixels with $(i = W - 2)$, such that $1 < j < H - 1$, the test will be done with only three pixels, $(i - 1, j)$, $(i - 1, j - 1)$, and $(i, j - 1)$ (Figure 21).

If one of the neighbors of the current pixel p belongs to some region R, and if the average gradient direction of R is close $\phi(p)$, then the pixel p should be added to that region R, as shown in Figure 22. If it is found that p does not belong to an existing gradient region, a new region will be created and the new pixel will be considered the first one in the new region. This situation is shown in Figure 23.

As we add a new pixel to an existing gradient region R, we maintain the region characteristics. Selecting one out of the neighboring pixels to join its region, if it belongs to any, is one of the most important tasks in this algorithm. The region of

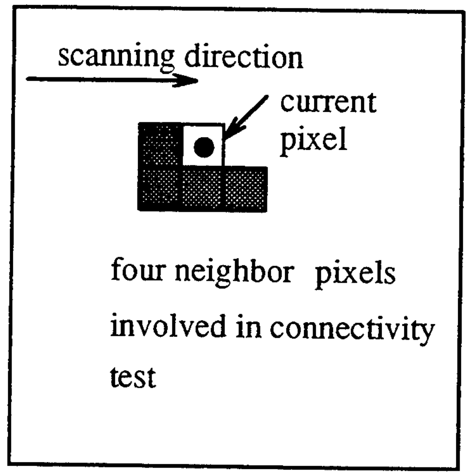


Figure 20. General case of connectivity test.

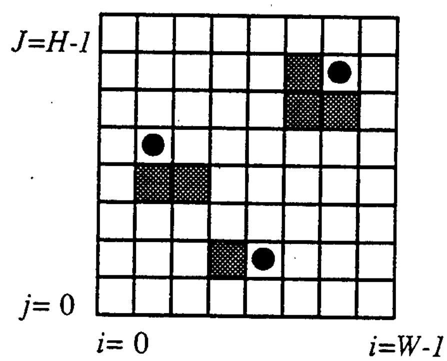


Figure 21. Special cases of connectivity test.

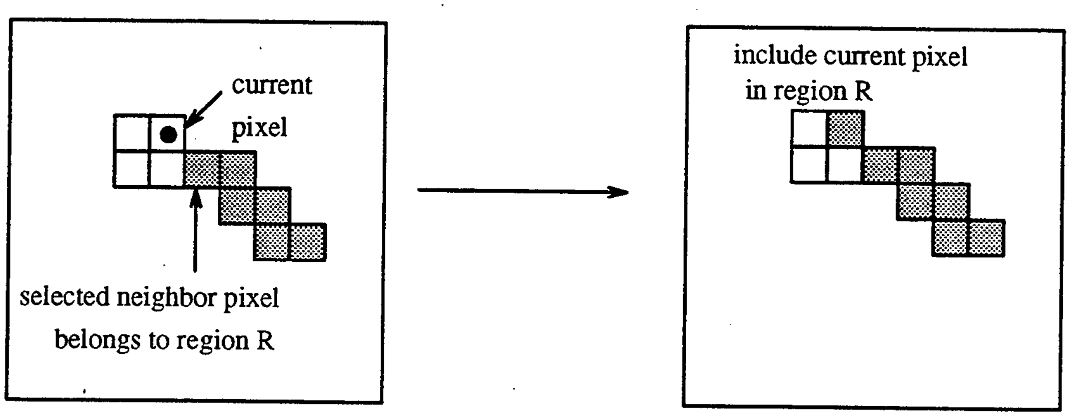


Figure 22. A pixel is included in the region as one of its neighbors if it satisfies the connectivity test.

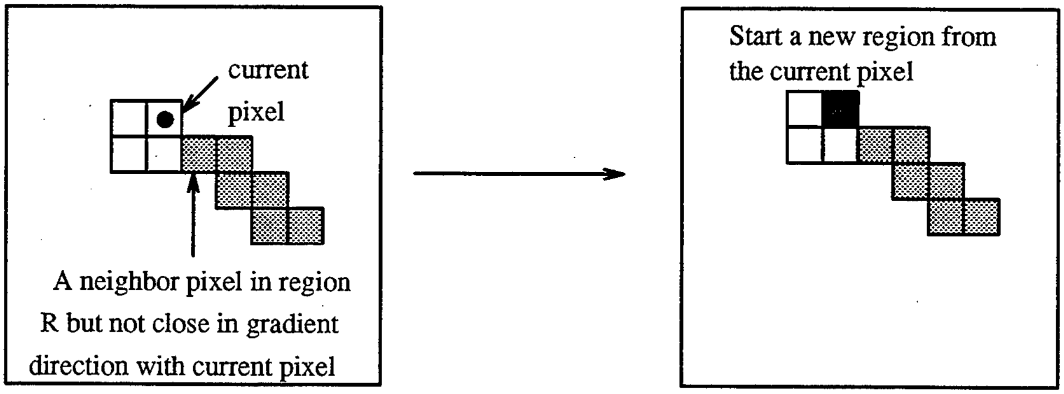


Figure 23. Current pixel starts a new region if it cannot belong to a region as one of its neighbors.

the chosen pixel should have the minimum difference in direction between its average gradient direction and that of the current pixel.

C. DATA STRUCTURES

We describe an image by a two-dimensional array of size $W \times H$. This corresponds to our usage of matrix representation of images, explained in the previous chapter. Each element represents the intensity value $f(p)$ of pixel $p = (x, y)$ with range of 0 to 255. The image pixel data structure is shown in Figure 24.

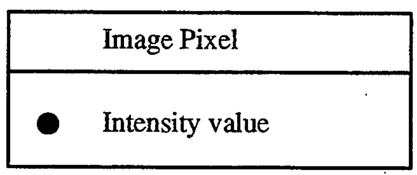


Figure 24. An image pixel data structure.

1. Row of Pixels

Two one-dimensional arrays $[0..W-1]$ of the same structure are used for storing processed pixel information on the current row in the image (Current) and the previous row (Previous). Each element of the array is indexed by the x-coordinate of the pixel and contains the information shown in Figure 25.

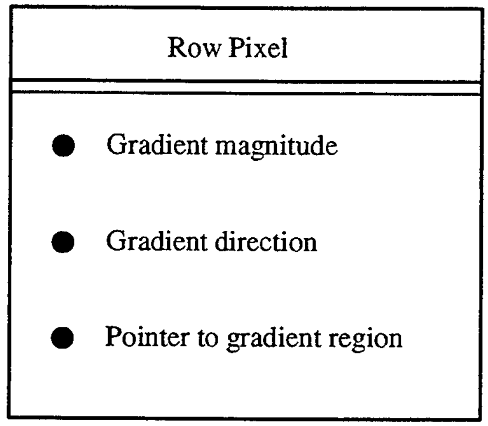


Figure 25. Current or previous row pixel data structure.

2. Gradient Regions

Gradient regions are described as a list structure (Figure 26). Each element of

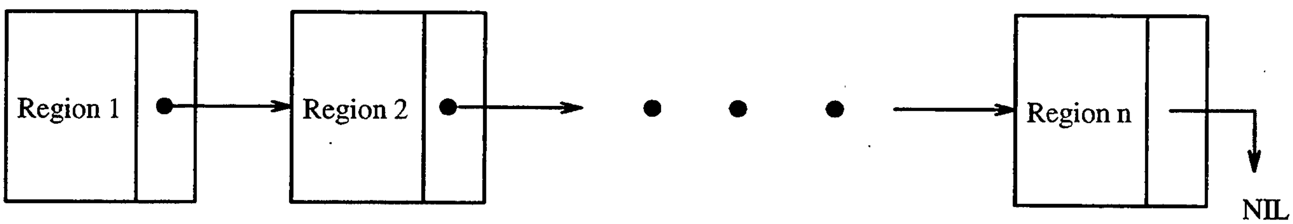


Figure 26. List structure of gradient regions.

the list corresponds to a gradient region and contains the moments and least-squares fitting parameters of the region and its first and last pixels. At the creation of a gradient region the parameters should be initialized, and they are updated when a new pixel is added to the region. From these parameters, the number of pixels, m_{00}, is used to judge the strength of a gradient region. A region with a very small number of pixels is interpreted as some noise in the image. It is desirable to have strong regions to obtain major line segments corresponding to them. Another attribute is the average value ϕ_{avg} of the gradient directions of the pixels included in the region. This attribute is used in the comparison to verify the closeness between a pixel and a region. To obtain the line segment, the moments $(m_{01}, m_{10}, m_{11}, m_{20}, m_{02})$ and centroid (μ_x, μ_y) are included in the structure. The first and last pixels (q_1 and q_2) of the region should be known. They are used to compute the exact endpoints (e_1 and e_2) of the line segment. The region structure is represented by the parameters shown in Figure 27.

Gradient Region

- Num. of pixels (m_{00})
- Sum of x-coordinates (m_{10})
- Sum of y-coordinates (m_{01})
- Sum of squares of x-coordinates (m_{20})
- Sum of squares of y-coordinates (m_{02})
- Sum of products of x and y-coordinates (m_{11})
- x-value of centroid (μ_x)
- y-value of centroid (μ_y)
- Starting pixel (q_1)
- End pixel (q_2)
- Gradient directions average (ϕ_{avg})
- Pointer to next region (Next)

Figure 27. Gradient region data structure.

3. Line Segment

A detected line segment L is represented by the parameters shown in Figure 28. These include the two endpoints e_1 and e_2, as well as the orientation α of the normal from the origin to L and the length r of the normal to L (See Chapter II). The number of pixels (m_{00}) of its gradient region also is included.

Line Segment

- Number of pixels (m_{00})
- First endpoint (e_1)
- Second endpoint (e_2)
- Length of its normal (r)
- Orientation of the normal (α)

Figure 28. Line segment data structure.

D. ALGORITHM

In this section, we describe the pseudo-code of the major algorithms of the method. Specifically, the main algorithm, the method to compute the gradient information at each pixel, the connectivity test, the least-squares fitting, and computation of the endpoints of the detected line segments will be discussed. The main algorithm is shown in Figure 29. The scanning is performed pixel-by-pixel (Line 1). The gradient magnitude g at pixel p is computed using the function ComputeGradientMagnitude$(p, \mathcal{I})$ in Line 2. If the computed value is larger than some threshold value G_t (Line 3), then p is considered *significant*, and hence the gradient direction ϕ will be computed in Line 4. Then the region R to which p will be included is returned as a result of performing the function ConnectivityTest(p, Current, Previous), in Line 5. In this function, we examine the neighboring pixels to p in the current and previous rows (as explained in Section B). One of two actions is performed based on the connectivity test. If a new region is created, it should be initialized with p as the first pixel (Lines 6-7). If p can be included in an already open region, we update the least-squares fitting parameters of the region and set the last pixel in R to be p. This is done through the function UpdateRegion(R, p) in Line 9. After the scanning process is completed, the next task is to scan through the list

```
DetectEdgesByScanning(I)
begin
1.  for each image pixel p ∈ I do
2.        g(p) = ComputeGradientMagnitude(p, I, g_x, g_y)
3.        if g(p) ≥ G_t then :
4.              φ(p) = atan2(g_y, g_x)
5.              ConnectivityTest(p,Current,Previous,R)
6.              if NewRegion(R)
7.                    InitializeRegion(R, p)
8.              else
9.                    UpdateRegion(R, p)
10. for each edge region R
11.       if  (m_00(R) ≥ n_0)
12.             LeastSquaresFitting(R, α, r)
13.             ComputeEndpoints(q_1(R), q_2(R), α, r, e_1, e_2)
14.             return ⟨m_00(R), α, r, e_1, e_2⟩
15.       else
16.             return nil
end
```

Figure 29. The main algorithm for edge detection by scanning.

of regions (Line 10). For any region that has a sufficient number of pixels (Line 11), we compute the segment parameters by the least-squares linear fitting, and the exact two endpoints (Lines 12-13) using the values of the first pixel in R, which we denote $q_1(R)$ and last pixel $q_2(R)$.

1. Computing Gradient Magnitude

The function **ComputeGradientMagnitude** is used to apply the Sobel operator at pixel $p = (x, y)$ for computing its gradient magnitude. The Sobel edge detector provides good performance and is relatively insensitive to noise. It provides a balance between the computation time and the accuracy of edge direction [Ref. 5]. The algorithm is presented in Figure 30. We use the two dimensional array $I[x][y]$ to represent the intensity values $f(p)$ for $p = (x, y)$.

ComputeGradientMagnitude$(p, \mathcal{I}, g_x, g_y)$
begin
1. $g_x = -I[x-1][y-1] - 2 \times I[x-1][y] - I[x-1][y+1]$
   $+I[x+1][y-1] + 2 \times I[x+1][y] + I[x+1][y+1]$
2. $g_y = -I[x-1][y-1] - 2 \times I[x][y-1] - I[x+1][y-1]$
   $+I[x-1][y+1] + 2 \times I[x][y+1] + I[x+1][y+1]$
3. return $\sqrt{g_x^2 + g_y^2}$

end

Figure 30. Computing gradient magnitude with Sobel operator.

2. Connectivity Test Function

The general algorithm of connectivity test is presented in Figure 31. In lines 1-8, the regions R_i of the neighboring pixels are determined and the differences in gradient directions d_i between $\phi(p)$ and the average gradient direction of each of R_i are computed. We take a region with the minimum difference d_i, if this value of d_i is less than some angle threshold or start a new region (Lines 9-14). Finally, the region R to which the pixel p belongs is determined and returned (Line 15).

3. Least-Squares Fitting

The least-squares fitting is a robust method to compute the parameters of a line segment that best fits a set of points in a region R. The parameters of the region are updated when a new pixel $p = (x, y)$ is added to R. The function **UpdateRegion**(R, p) performs this task. Now the values of r and α for the normal to the required line segment can be obtained. They are computed by the function **LeastSquaresFitting**, which is presented in Figure 32. Notice that the moments $m_{00}, m_{10}, m_{11}, m_{20}$, and m_{02} are among the parameters of region R. For simplicity, we use them without referring to R in the algorithm.

ConnectivityTest(p,Current,Previous,R)
begin
$R_1 = \text{Region}(\text{Current}[x-1])$
$d_1 = \text{Normalize}(|\phi(p) - \phi_{avg}(R_1)|)$
$R_2 = \text{Region}(\text{Previous}[x-1])$
$d_2 = \text{Normalize}(|\phi(p) - \phi_{avg}(R_2)|)$
$R_3 = \text{Region}(\text{Previous}[x])$
$d_3 = \text{Normalize}(|\phi(p) - \phi_{avg}(R_3)|)$
$R_4 = \text{Region}(\text{Previous}[x+1])$
$d_4 = \text{Normalize}(|\phi(p) - \phi_{avg}(R_4)|)$
$(d, i) = \text{MinValueAndIndex}(d_1, d_2, d_3, d_4)$
if $d > \delta$; no closeness in gradient directions
    $R = \text{CreateRegion}()$
    $\text{NewRegion}(R) = \text{TRUE}$
else
    $R = R_i$
return R
end

Figure 31. General connectivity test algorithm.

LeastSquaresFitting(R, α, r)
begin
$M_{20} = m_{20} - \frac{m_{10}^2}{m_{00}} m_{10}^2$
$M_{11} = m_{11} - \frac{m_{10} m_{01}}{m_{00}}$
$M_{02} = m_{02} - \frac{m_{01}^2}{m_{00}}$
$\alpha = 0.5 \times \text{atan2}(-2M_{11},\ M_{02} - M_{20})$
$r = \mu_x \times \cos(\alpha) + \mu_y \times \sin(\alpha)$
return $\langle \alpha, r \rangle$
end

Figure 32. Least-squares fitting of a region.

ComputeEndPoints$(q_1, q_2, \alpha, r, e_1, e_2)$
begin
1. $\delta_1 = x_{q1} \times cos(\alpha) + y_{q1} \times sin(\alpha) - r$
2. $\delta_2 = x_{q2} \times cos(\alpha) + y_{q2} \times sin(\alpha) - r$
3. $x_1 = x_{q1} - \delta_1 \times cos(\alpha)$
4. $y_1 = y_{q1} - \delta_1 \times sin(\alpha)$
5. $x_2 = x_{q2} - \delta_2 \times cos(\alpha)$
6. $y_2 = y_{q2} - \delta_2 \times sin(\alpha)$
7. **return** $\langle e_1 = (x_1, y_1), e_2 = (x_2, y_2) \rangle$

end

Figure 33. Endpoints computation algorithm.

4. Computing Endpoints of Segments

After computing the least-squares parameters (r, α), we compute the two endpoints $e_1 = (x_1, y_1)$ and $e_2 = (x_2, y_2)$, using the first pixel $q_1 = (x_{q1}, y_{q1})$ and last pixel $q_2 = (x_{q2}, y_{q2})$. This is described in Figure 33.

E. EXPERIMENTAL RESULTS

The algorithm was tested on actual images taken by a CCD camera. The input images shown in Figures 34 and 36 are used to produce the line segments shown in Figures 35 and 37, respectively, using the edge detection algorithm described.

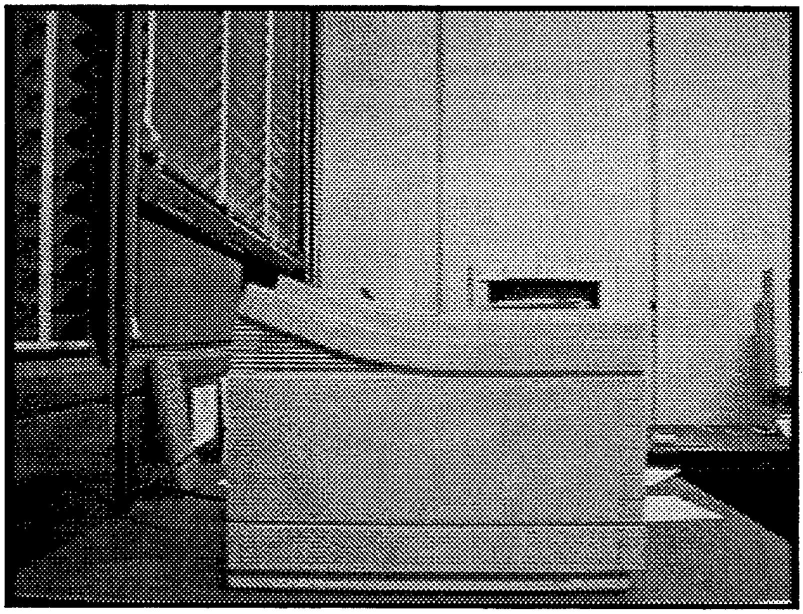


Figure 34. Input image of a printer.

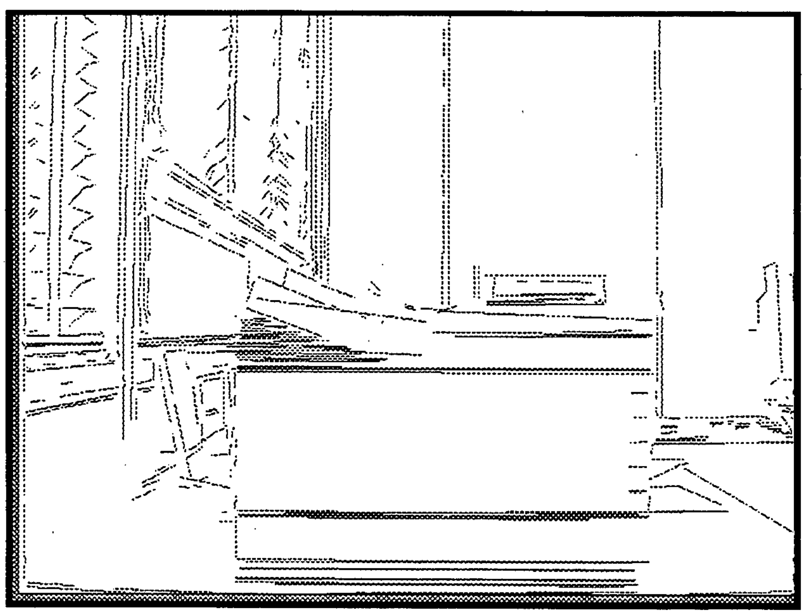


Figure 35. Line segments extracted from a printer image.

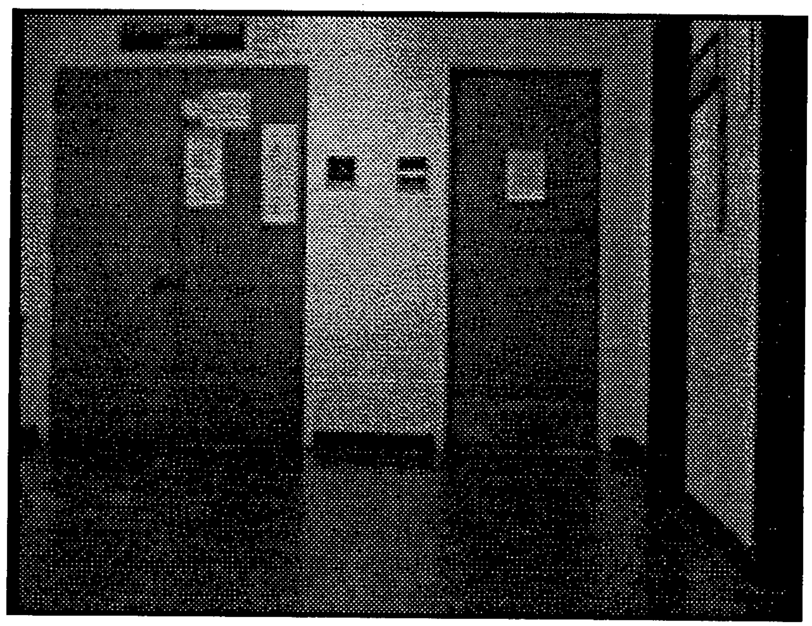


Figure 36. Input image of a hallway portion.

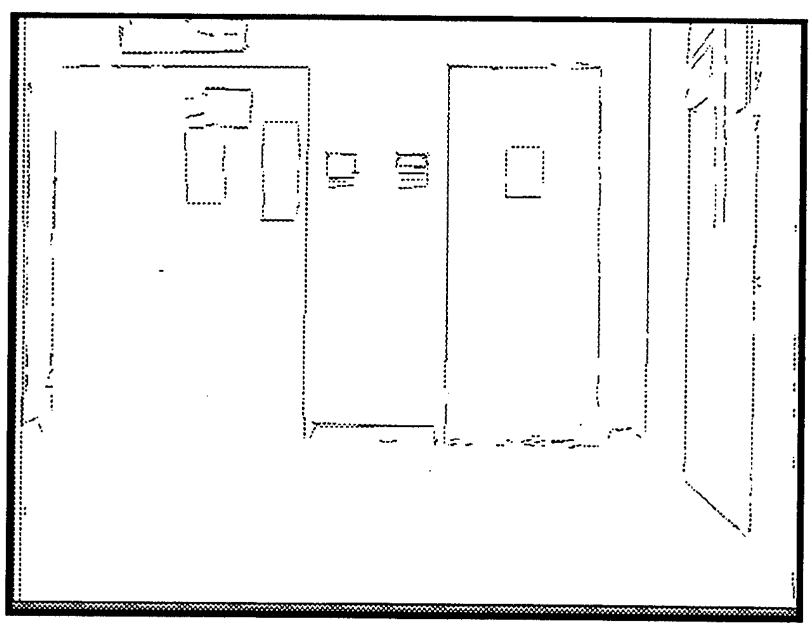


Figure 37. Line segments detected form the hallway portion image.

IV. DIRECTION-CONTROLLED EDGE TRACKING METHOD

A. PRINCIPLE

In this chapter we present a new method of detecting an edge, called "Direction-Controlled Edge Tracking" [Ref. 70]. Given a significant pixel p_0 in an image, the edge tracking task is to find a sequence Q of pixels on the edge containing p_0. This task is performed in two opposite directions (ψ_+ and ψ_-) that are approximately orthogonal to the gradient direction $\phi(p_0)$, as shown in Figure 38. To illustrate the

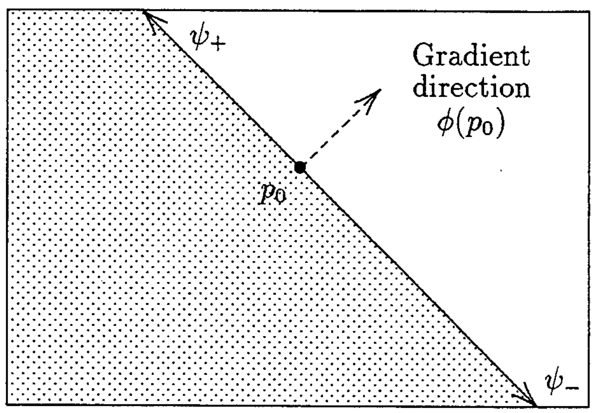


Figure 38. Edge tracking in two opposite directions.

idea of the algorithm, Figure 39 shows the sequence of pixels obtained by tracking the slanted edge in the lower right portion of the image shown in Figure 36 in the previous chapter. The best-fit line segment is also shown. The best-fit line L to Q is computed through the least-squares fitting method to obtain (i) the normal direction α, and (ii) the distance r from the origin to the line L. However, finding the equation of this line is not enough for applications in image understanding which require computing the endpoints of the line segments. While tracking a pixel sequence Q, the two end pixels of this sequence, q_1 and q_2, are logged. Using these two pixels as the best estimate of the edge endpoints is not recommended, because noise may affect their

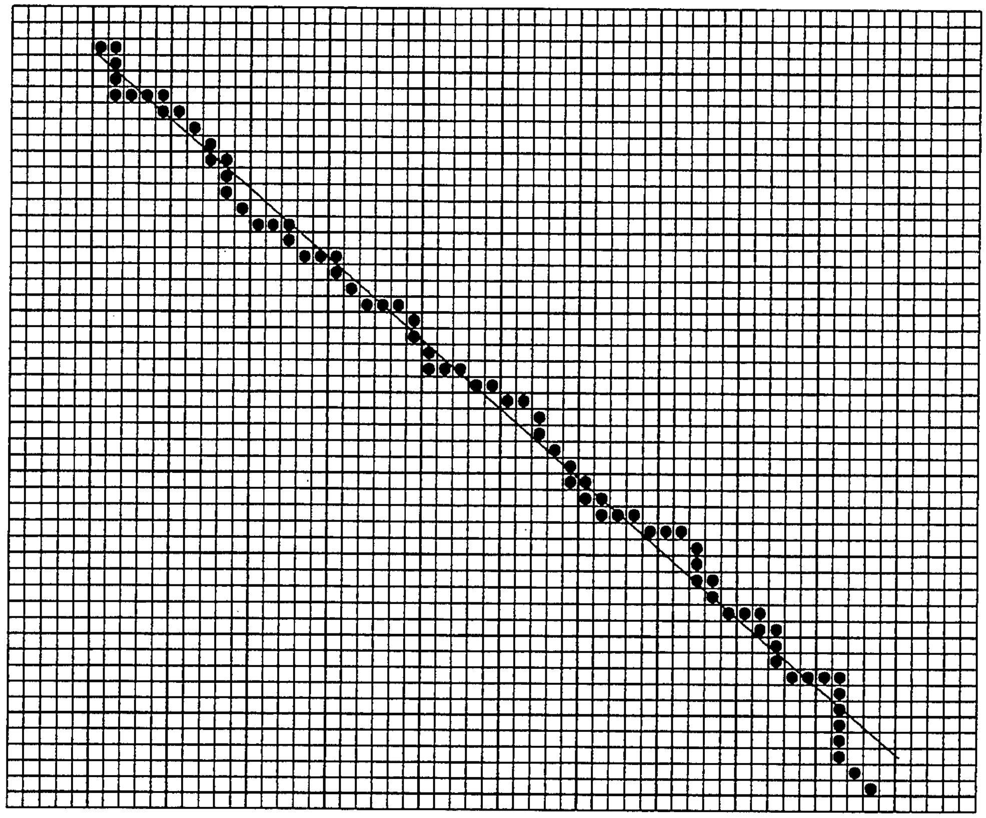


Figure 39. Edge pixel sequence with its fitted line segment.

positions. Instead, q_1 and q_2 are used to compute the two projected endpoints e_1 and e_2 as explained later. We call a straight line L corresponding to a pixel sequence Q *major* when the number of pixels n contained in Q is more than a threshold value n_0, say 50.

1. Edge Tracking Algorithm

Given an image I and an initial pixel p_0, the algorithm returns a line segment L corresponding to an edge on the image (Figure 40). The overall algorithm for edge tracking and segment detection is described in Figure 41. The function "TrackPixels" in lines 2 and 4 is further elaborated in Figure 43.

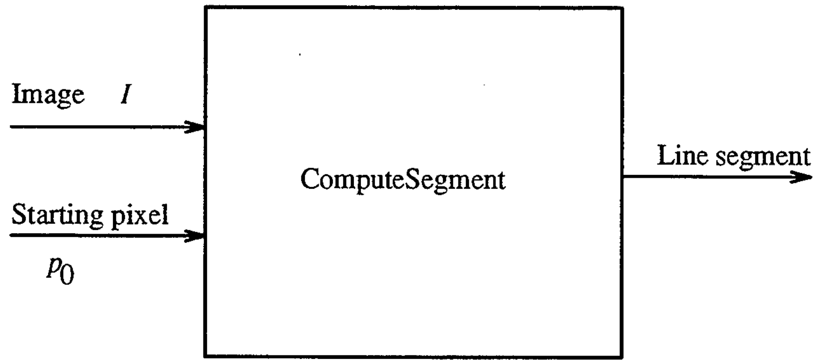


Figure 40. Line segment detection by tracking.

B. TRACKING CONTROL BY EDGE DIRECTION

The edge tracking task is performed as a sequence of **SelectNextPixel** operations which select a neighboring pixel p_2 from the current pixel p_1. In each SelectNextPixel operation, selecting the neighboring pixel with the maximum gradient magnitude is apparently a reasonable strategy, since this criterion minimizes a possible noise effect. However, if an edge is tracked under this criterion only, a curved line or a sequence of connected line segments might be tracked. Such a result does

ComputeSegment$(p_0, \mathcal{I})$
begin
$Q = \{p_0\}$
$Q = \text{TrackPixels}(p_0, Q, \mathcal{I}, \frac{\pi}{2})$
$q_1 = \text{LastPixel}(Q)$
$Q = \text{TrackPixels}(p_0, Q, \mathcal{I}, -\frac{\pi}{2})$
$q_2 = \text{LastPixel}(Q)$
$\text{LeastSquaresFitting}(Q, \alpha, r,)$
$\text{ComputeEndPoints}(q_1, q_2, \alpha, r, e_1, e_2)$
if $(|Q| \geq n_0)$
    return $\langle \alpha, r, e_1, e_2 \rangle$
else
    return nil
end

Figure 41. Edge tracking and segment detection algorithm.

not meet the requirement of detecting straight edges separately. To avoid this uncontrolled tracking, the direction ψ of the segment being tracked is computed using the partial Q obtained so far. This direction ψ is computed at each step and is used to control the SelectNextPixel process in the following way: (i) ψ defines three neighbor pixels from which the next pixel will be selected, and (ii) a selected pixel is tested for the consistency between its gradient direction and ψ.

1. Tracking Direction Evaluation

To obtain robust orientation information ψ for Q, the least-squares fitting algorithm is applied to Q. By doing this, we obtain the normal direction α of the best fitted line for it (assuming $|Q| \geq 2$). Since the direction α is computed by the positional information of all the pixels collected so far, this value is expected to contain extremely fine directional information. Notice that the range of α is $[-\frac{\pi}{2}, \frac{\pi}{2}]$. We observed in Figure 38 that all pixels in a given straight edge must have a similar gradient direction ϕ, and this value should be close to α or $\pi - \alpha$. From this observation,

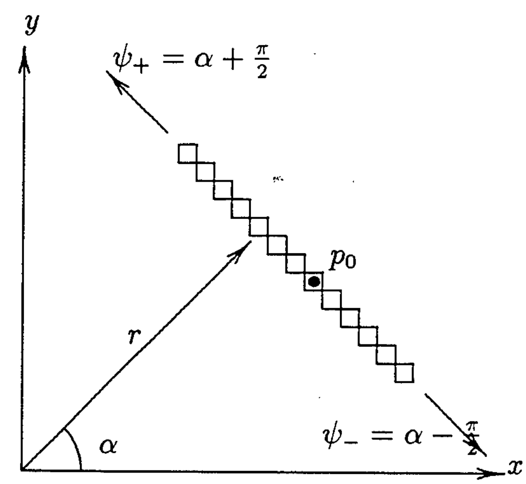


Figure 42. Finding tracking direction.

we assume that the direction ψ of an expected edge is perpendicular to α. Therefore, we compute ψ_+ and ψ_- as

$$\psi_+ = \alpha + \frac{\pi}{2},$$

$$\psi_- = \alpha - \tfrac{\pi}{2}.$$

This relation is shown in Figure 42. Obviously, ψ_+ and ψ_- are mutually apart by π. The edge is first tracked in the direction of ψ_+ in the range of $[0, \pi]$, and next it is tracked in the opposite direction ψ_- in the range of $[-\pi, 0]$ to complete the edge. However, in the initial state where $Q = \{p_0\}$, it is impossible to find α by least-squares fitting. In this case, the gradient direction $\phi(p_0)$ is used to evaluate $\psi_\pm$. Since the range of $\phi(p_0)$ is $[-\pi, \pi]$, we need to normalize it into the interval of $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ to keep consistency in the evaluation of $\psi_\pm$.

$$\psi_\pm = \mathcal{N}(\phi(p_0)) \pm \frac{\pi}{2}.$$

The normalization function $\mathcal{N}(\gamma)$ for an angle γ is defined as follows:

$$\mathcal{N}(\gamma) = \begin{cases} \mathcal{N}(\gamma - \pi), & \text{if } \gamma > \frac{\pi}{2} \\ \mathcal{N}(\gamma + \pi), & \text{if } \gamma < -\frac{\pi}{2} \\ \gamma, & \text{if } -\frac{\pi}{2} \le \gamma \le \frac{\pi}{2} \end{cases}.$$

For instance, $\mathcal{N}(\frac{\pi}{4}) = \mathcal{N}(\frac{5\pi}{4}) = \mathcal{N}(-\frac{3\pi}{4}) = \frac{\pi}{4}$.

2. One-Way Pixel Sequence Tracking Algorithm

Figure 43 shows the algorithm of tracking a pixel sequence Q starting with a pixel p_0 in one way. As mentioned earlier, this procedure is performed twice, once in the direction ψ_+ (with $\beta = \frac{\pi}{2}$) and next in the direction ψ_- (with $\beta = -\frac{\pi}{2}$), as shown in the main edge tracking algorithm (Figure 41). The moment calculation for the least-squares fitting task is included in lines 4 and 12.

C. SELECTING NEXT PIXEL

1. Quantization of Edge Direction

As outlined in the previous subsection, the edge direction ψ (ψ_+ or ψ_-) is utilized in two ways:

1. Finding which specific pixels among eight neighbors should be examined for selecting a next pixel, and

```
TrackPixels(p0, Q, I, β)
begin
if |Q| = 1
    α = N(φ(p0))
else
    LeastSquaresFitting(Q, α, r)
p1 = p0
doforever;
    ψ = α + β
    p2 = SelectNextPixel(p1, ψ, I)
    if (p2 = nil)
        exit
    Q = Q ∪ {p2}
    LeastSquaresFitting(Q, α, r)
    p1 = p2
return Q
end
```

Figure 43. Tracking one side of the pixel sequence.

2. Testing whether a pixel among these neighbors has a gradient direction consistent with ψ.

First, the domain $[-\pi, \pi]$ of ψ is divided into eight sub-domains

$$\mathcal{D}_i = \left[\frac{2i-1}{8}\pi, \frac{2i+1}{8}\pi\right] \quad \text{for } i = 0, \cdots, 7.$$

For instance, $\psi = \frac{\pi}{4}$ belongs to $\mathcal{D}_1 = \left[\frac{1}{8}\pi, \frac{3}{8}\pi\right]$. For each sub-domain $\mathcal{D}_i$, we assign a set $\mathcal{S}_i$ of three neighboring pixels, as shown in Figure 44. For example, if the current pixel is $p_1 = (x, y)$, and if $\psi \in \mathcal{D}_1$, the next pixel must be selected from $\mathcal{S}_1 = \{(x, y+1), (x+1, y+1), (x+1, y)\}$, as shown at the top right portion in Figure 44. The pixel $(x+1, y+1)$ is precisely located in the direction $\frac{\pi}{4}$ from the pixel p_1. The set $\mathcal{S}_1$ includes two more neighboring pixels of $(x+1, y+1)$.

2. Next Pixel Selection

Given the set $\mathcal{S}_i$, the selection of a next pixel p_2 is performed as follows:

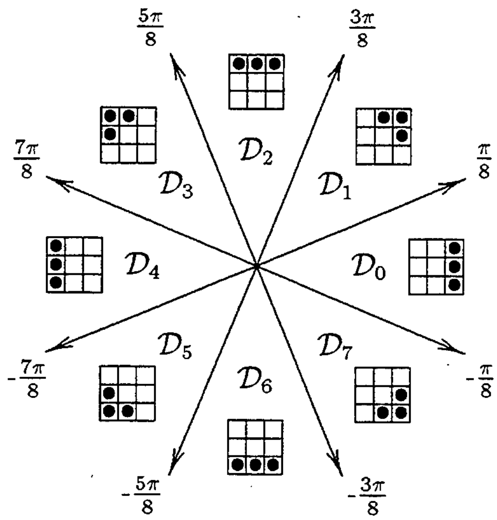


Figure 44. Next pixel set for each range $\mathcal{D}_i$.

- We choose the most significant pixel p_2 in $\mathcal{S}_i$, and
- Test whether its gradient direction $\phi(p_2)$ and the tracking direction ψ are *consistent*, i.e., $\phi(p_2)$ is approximately normal to ψ.

Definition: Two directions ϕ and ψ are considered to be consistent if

$$\left|\mathcal{N}\left(\phi - \psi - \frac{\pi}{2}\right)\right| < \epsilon,$$

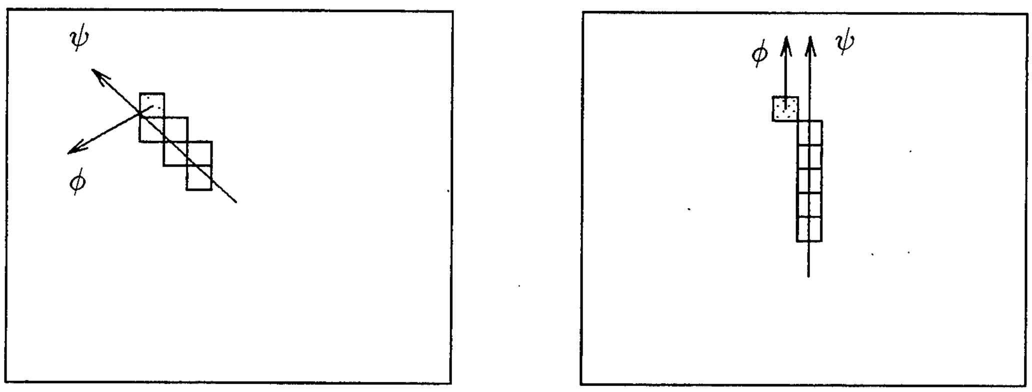
for a small angle threshold ϵ.

If the most significant pixel passes this test, it is defined to be the next pixel p_2. If this consistency condition is not satisfied for the most significant pixel, then we test whether there is another significant pixel in $\mathcal{S}_i$. If so, the consistency test is executed again for this pixel. If this is consistent, this becomes the next pixel p_2. Otherwise, the test is executed for the last (third) pixel in $\mathcal{S}_i$. In other words, the selection and consistency test for the remaining pixels in $\mathcal{S}_i$ are repeated. Figure 46 shows two cases: (a) two directions ψ and ϕ are consistent, or (b) they are inconsistent.

Suppose the next pixel p_2 is found. In this case, this new pixel will be added to the pixel sequence Q and is defined as the current pixel p_1. From that pixel, the SelectNextPixel operation will be repeated again. If there is no pixel that satisfies both conditions, the result **nil** is returned when one side of tracking task ends in this

```
SelectNextPixel(p1, ψ, I)
begin
    S = Neighbors(p1, ψ)
    while S ≠ ∅
        p2 = LargestGradientPixel(S, I)
        if Consistent(ψ, φ(p2))
            return p2
        S = S − {p2}
    return nil
end
```

Figure 45. Finding next pixel.



(a) Two consistent directions (b) Inconsistent directions

Figure 46. Consistency test.

specific direction (ψ_+ or ψ_-). The description of the SelectNextPixel function is given in Figure 45.

D. COMPUTING SEGMENT PARAMETERS

The least-squares fitting method provides the representation (r, α) of the line segment that corresponds to the detected edge. However, some other information may be needed for further processing such as m_{00}, the number of pixels tracked for the edge. For many applications including image understanding, an estimate of the two endpoints of the line representing Q is also essential. For that purpose, we keep a record of the first and last pixels $(q_1, q_2) = (p_1, p_n) = \big((x_{q1}, y_{q1}), (x_{q2}, y_{q2})\big)$ in the sequence Q. We use their projection points (e_1, e_2) on the line [Ref. 69]. They are computed as

$$e_k = (x_{qk} - \delta_k \cos\alpha,\ y_{qk} - \delta_k \sin\alpha), \quad \text{for } k = 1, 2,$$

where

$$\delta_k = x_{qk}\cos\alpha + y_{qk}\sin\alpha - r, \quad \text{for } k = 1, 2.$$

V. A GLOBAL ALGORITHM FOR EDGE DETECTION USING RANDOM HITTING

To detect all major edges in an image, we want to use the edge tracking algorithm described in the previous chapter. To obtain initial pixels to start edge tracking and to minimize processing of insignificant pixels, we introduce the use of a *random-hitting* method.

The global algorithm for straight-edge detection is designed as shown in Figure 47. A random position is computed for an initial pixel p_0 (Line 3). If it is significant and is not close to any of the previously detected line segments (Line 4), the edge tracking and segment detection algorithm (Figure 41) is executed starting from p_0. If the new line segment L is a major one, it is added to the set $\mathcal{L}$ of detected major line segments. This process is repeated until K line segments are obtained (Line 2).

DetectEdges($\mathcal{I}$,K,$\mathcal{L}$)
Input : Image: $\mathcal{I}$, estimated number of lines : K
Output : Set of line segments: $\mathcal{L}$
begin
1. $\mathcal{L} = \emptyset$
2. **while** $|\mathcal{L}| < K$ **do**
3. p_0 = RandomPosition()
4.   **if** $\Big($ Significant$(p_0) \wedge$ NotClose$(p_0, \mathcal{L})$ $\Big)$
5. L= ComputeSegment$(p_0, \mathcal{I})$
6.   **if** Major(L)
7. $\mathcal{L} = \mathcal{L} \cup L$
end

Figure 47. Global algorithm for fast straight edge detection.

A. RANDOM HITTING

1. Concept of Random Hitting

Although the most straightforward approach for detecting all edges in an image is to examine all pixels, most of the pixels in a normal image are insignificant. Therefore, this method is extremely inefficient. To minimize the number of examinations of insignificant pixels, we compute (hit) a pixel position p_0 randomly. If p_0 is significant, the edge tracking algorithm, explained in the previous chapter, is executed. We adopt the linear congruential pseudo-random integer generating method [Ref. 71]. A pseudo-random number sequence $\langle R_n \rangle$ is defined as follows:

$$R_{n+1} = (aR_n + c) \bmod M, \qquad\qquad n \geq 0, \tag{V.1}$$

where M is the modulus ($M > 0$), a the multiplier ($0 \leq a < M$), c the increment ($0 \leq c < M$), and R_0 the starting value ($0 \leq R_0 < M$).

A desirable property for a pseudo-random number generator is that no integer will be generated twice before the sequence of the pseudo-random numbers includes all non-negative integers less than M. The following theorem in [Ref. 71] describes a necessary and sufficient condition to the parameters a and c to satisfy this property.

Theorem V.1 *The linear congruential sequence (V.1) defined by M, a, c, and R_0 has a period length M if and only if*
i) c is relatively prime to M;
ii) $b = a - 1$ is a multiple of p, for every prime p dividing M;
iii) b is a multiple of 4, if M is a multiple of 4.

Recall that two integers a, b are said to be *relatively prime* if their greatest common divisor is 1.

From now on, we let a=1 and let c be a number relatively prime to M to achieve the maximum period length of M. Furthermore, we apply this pseudo-random number method to compute a sequence of pixel positions for an image of $W \times H = M$ pixels. Each pseudo-random number $R = R_i$ is then converted into a pixel position $p = (x, y)$ by the transformation

$$p = (x, y) = (R \bmod W,\ R \div W),$$

where W is the width, in pixels, of an image. Then obviously a distinct pair of random numbers R_i and R_j are transformed into distinct pixel positions p_i and p_j, and vice versa.

Because it is decided that $a = 1$, the next and last decision is on the selection of c. To further simplify the choice of c, we propose to choose two numbers j and k, with $(0 < j < H)$ and $(0 < k < W)$, where k is relatively prime to W. Then the value of c is given by

$$c = j \times W + k.$$

We applied this random hitting method to our edge-detection experiments. In our camera system,

$$M = W \times H = 646 \times 486 = (2 \times 17 \times 19) \times (2 \times 3^5) = 2^2 \times 3^5 \times 17 \times 19 = 313,956.$$

Specifically, we chose $j = 238$ and $k = 181$, then

$$c = j \times W + k = 238 \times 646 + 181 = 182,999.$$

To illustrate the concept, in Figure 48 the first ten pixels are shown with the order in which they are hit using this pseudo-random number scheme. The distribution of 3000 pixels is shown in Figure 49.

Another desirable property for this pseudo-random hitting scheme is that, if we divide an image into unit areas with the same size, the probability of being hit is approximately equal for each of them at any time. We found that the previous choice of c looks as if it satisfies this property of *uniform distribution.* This pseudo-random number scheme brought the experimental results presented in Section C.

2. Edge Hitting Probability

We discuss a theoretical analysis for soundness of adopting random hitting to perform edge detection. We consider an image $\mathcal{I}$ with a total of M pixels. How many hits are needed to detect a specific small region containing m pixels in $\mathcal{I}$ (Figure 50)? More precisely, what is the probability for the region to be detected after n hits?

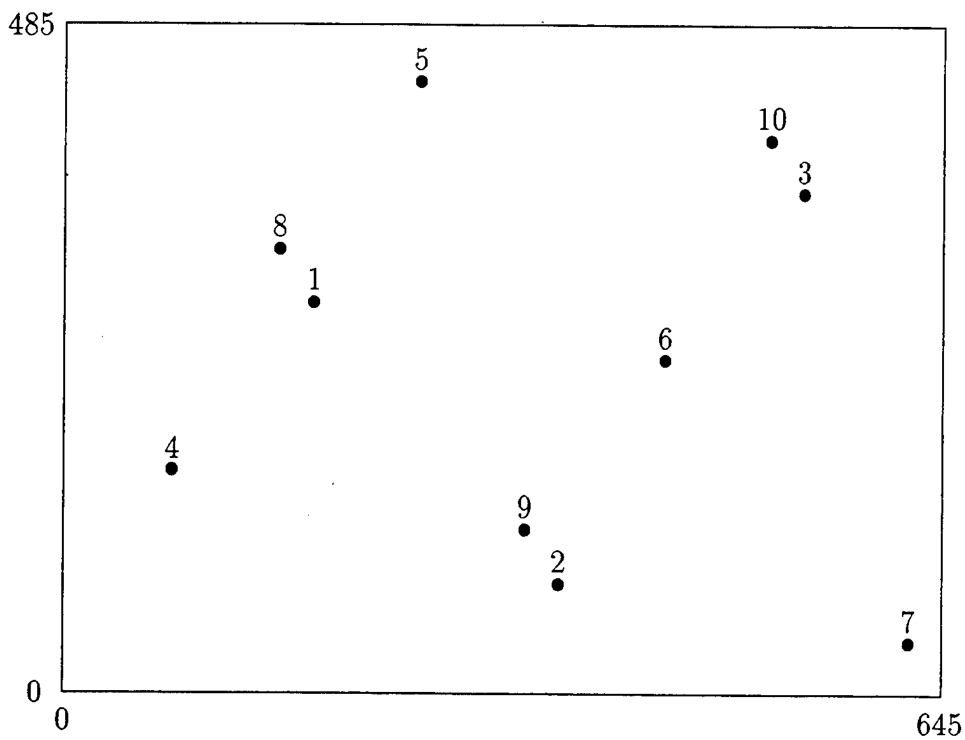


Figure 48. Distribution of first 10 hits using pseudo-random number generator with $c = 283 \times 646 + 181 = 182,999$.

First we compute the function $f(n)$, the probability for the region *not* being detected in n random hits. This "missing probability function" $f(n)$ is computed as:

$$f(n) = \left(\frac{M-m}{M}\right)^n = \left(1 - \frac{m}{M}\right)^n.$$

Let $Z = \frac{M}{m}$. Then

$$f(n) = \left(1 - \frac{1}{Z}\right)^n.$$

This is obviously a function of n. Specifically, let $n = Z$. Then

$$f(Z) = \left(1 - \frac{1}{Z}\right)^Z.$$

It is well-known that

$$\lim_{h \to \infty} f(h) = \frac{1}{e},$$

where e is the basis of natural logarithm. Since m is a positive integer, the value of Z is always finite and $f(Z)$ is never precisely equal to $\frac{1}{e}$. However, for a realistic

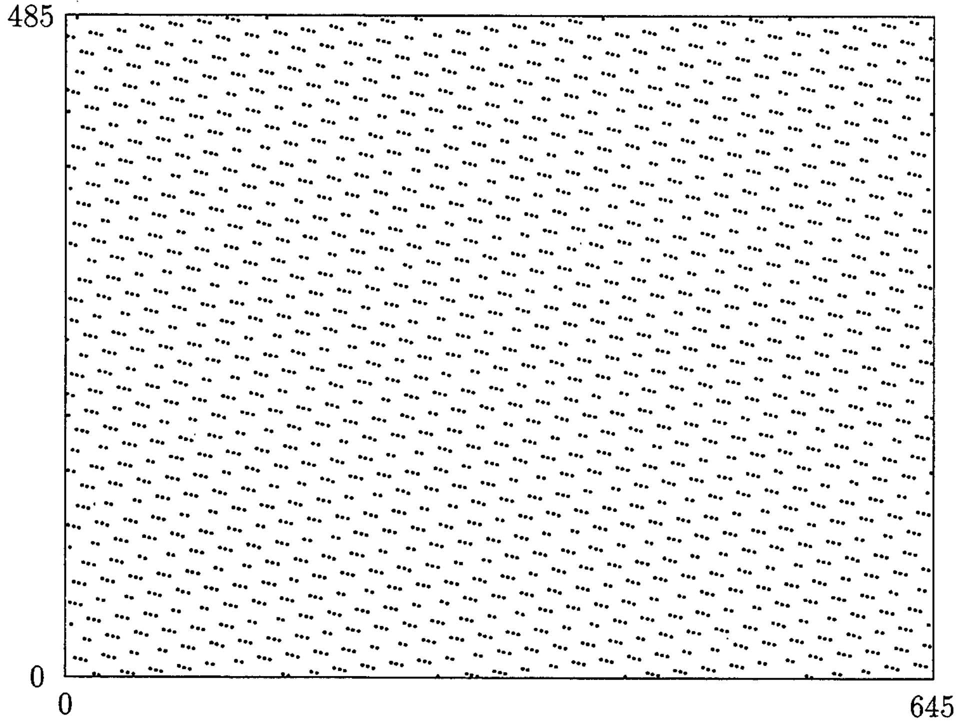


Figure 49. Distribution of 3000 pixels using pseudo-random number generation with $c = 283 \times 646 + 181 = 182,999$.

value of $Z = \frac{M}{m}$, $f(Z)$ is very close to $\frac{1}{e}$. For instance, if $Z= 100$, the relative error between $f(100)$ and $\frac{1}{e}$ is approximately 0.5 % and the one between $f(1000)$ and $\frac{1}{e}$ is approximately 0.05 %. Thus, we can use $\frac{1}{e}$ as a good approximation for $f(Z)$ for a large value of Z.

Now we explain how this analysis is related to the random hitting algorithm. Suppose we want to find an edge with a minimum length of 50 pixels . A region containing such an edge includes at least m=100 pixels. Assuming an image size of M=300,000 pixels and $Z = \frac{M}{m}$=3000 times, then the missing probability function $f(3000) \approx \frac{1}{e} = 0.36788$, and hence the detection probability is approximately 0.63212. Furthermore, if $n = 2Z = 6000$, then

$$f(2Z) = \left(1 - \frac{1}{Z}\right)^{2Z} = \left(\left(1 - \frac{1}{Z}\right)^{Z}\right)^{2} \approx \left(\frac{1}{e}\right)^{2} \approx 0.135.$$

That increases the detection probability to 0.865. Even with $3Z = 9000$ which is only 3% of the total number of pixels, the detection probability is about 95%. Table I

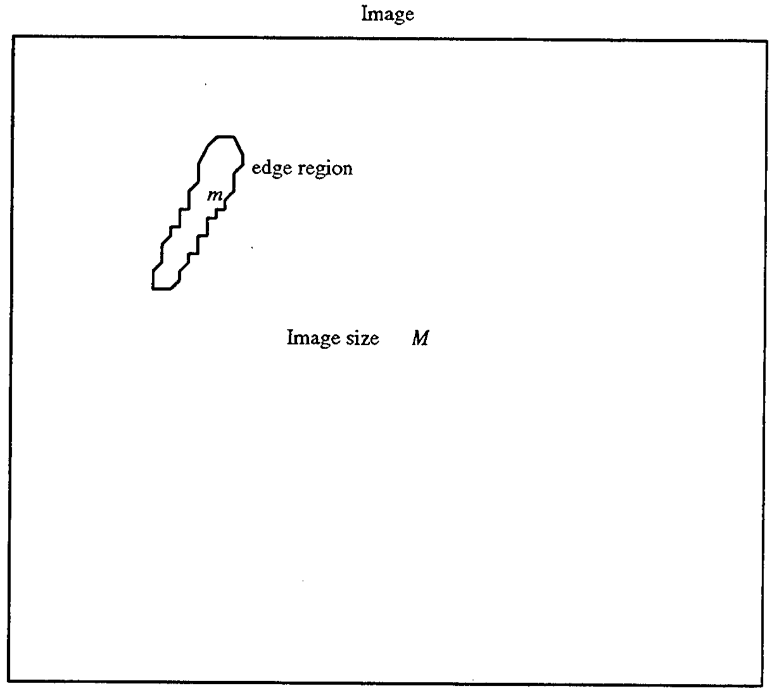


Figure 50. An Image with total of M pixels having an m-pixels region.

shows the approximation of detection probabilities for $n = Z$, $2Z$, $3Z$, $4Z$, and $5Z$.

n	missing probability	detection probability
Z	0.367	0.633
2Z	0.135	0.865
3Z	0.049	0.951
4Z	0.018	0.982
5Z	0.007	0.993

Table I. Approximation of detection probabilities for different n hits.

With this analysis, for k regions in the image, with m_1, m_2, $\cdots$, m_k pixels each, (Figure 51), let

$$m = \min_i m_i.$$

Then, using only $3 \times \frac{M}{m}$ hits, almost all the segments will be detected. In the examples of results in this chapter, Section C, we tried to detect most of the major segments with at least 50 pixels length ($m = 100$) from a test image of size 313,956. After only $2960 = 0{,}94\ Z$ hits, 20 of an expected 24 of those major edges were detected.

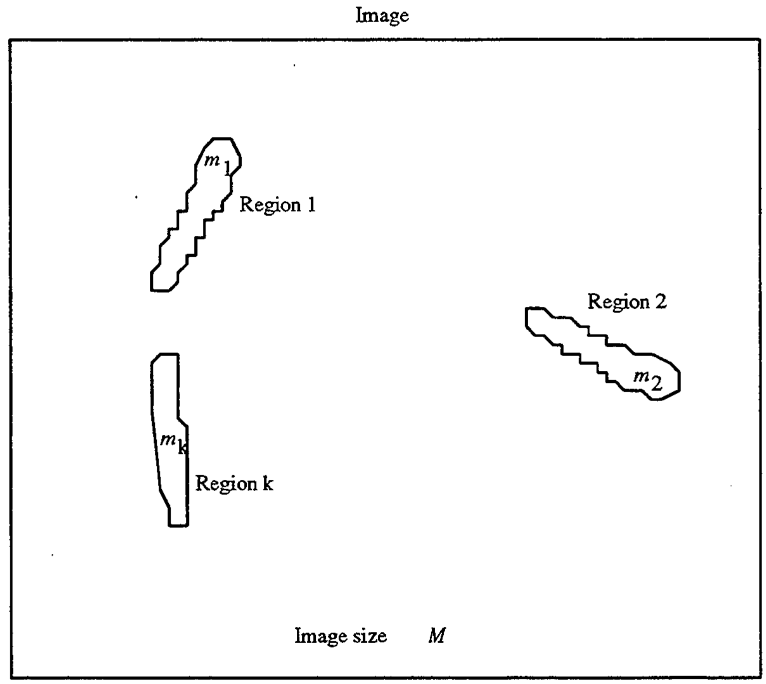


Figure 51. An Image with total of M pixels having different regions.

B. CLOSENESS TEST

In this global edge detection algorithm, a randomly generated pixel p_0 becomes a "seed" for a new segment if p_0 is significant. However, when some line segments have already been detected, the newly generated pixel p_0 may lie on one of these segments. We should avoid blindly tracking the same segment again starting from this pixel p_0. Therefore, each randomly generated pixel p_0 must be examined to determine whether it is close to any of the previously detected segments. For this test, the distance $d(p_0, L)$ between a given point p_0 and a given line segment should be computed. A segment is represented as

$$L = \overline{e_1 e_2} = \overline{(x_1, y_1)\ (x_2, y_2)},$$

where e_1 and e_2 are its endpoints with $e_1 \neq e_2$. Given a segment L, the plane is divided into three regions (Figure 52); V_1, V_2, and V_0. V_1 is the set of points p where e_1 is the closest point on L from p. Likewise, V_2 is the set of points p such that e_2 is the closest point on L from p. V_0 is the set of points p such that the closest point

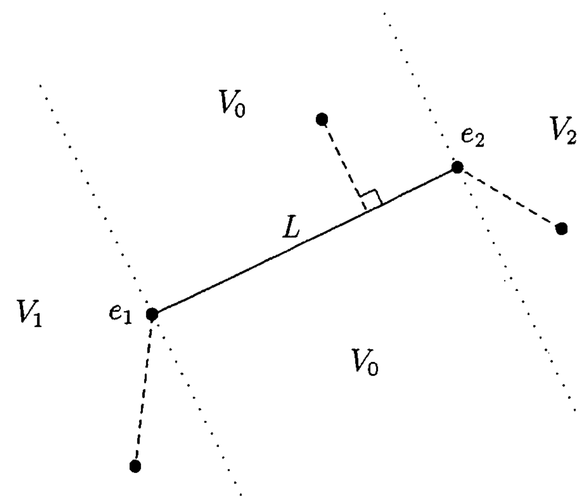


Figure 52. Finding distance from point to line.

on L from p is neither e_1 nor e_2. Given p, if we know the region V_i containing p, the distance $d(p, L)$ is easily obtained. The 2D transformation group theory [Ref. 72] is applied to solve this problem of determining V_i.

With the segment L, a *local* coordinate system is defined as follows. One of the two endpoints, say e_1, is considered as origin O_L (Figure 53). Its local X-axis (x_L) is aligned to the segment L, and the orientation θ of x_L is

$$\theta = \mathrm{atan2}(y_2 - y_1, x_2 - x_1).$$

Thus, the L-coordinate system is represented as $q_L \equiv (x_1, y_1, \theta)^T$ in the *global* image coordinate system. The pixel position $p = (x, y)$ in the global coordinate system is converted into the L-coordinates (x^*, y^*) as follows [Ref. 72]:

$$x^* = (x - x_1)\cos\theta + (y - y_1)\sin\theta \tag{V.2}$$

$$y^* = -(x - x_1)\sin\theta + (y - y_1)\cos\theta. \tag{V.3}$$

The local x-coordinate of the other endpoint e_2 is, using Equation (V.2),

$$x_2{}^* = (x_2 - x_1)\cos\theta + (y_2 - y_1)\sin\theta. \tag{V.4}$$

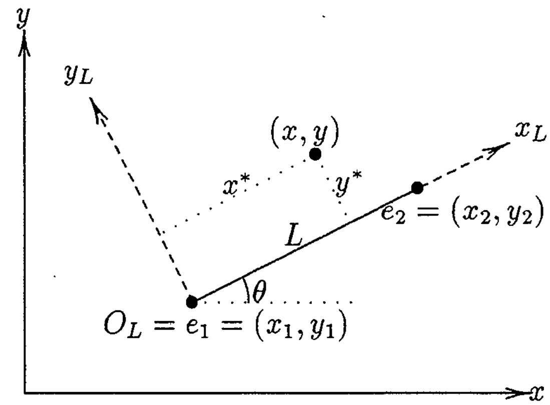


Figure 53. Pixel position in the L-coordinates

By transforming (x, y) in the image coordinates into (x^*, y^*) in the L-coordinate system, the distance $d(p, L)$ can be easily computed as

$$d(p, L) = \begin{cases} d(p, e_1) & \text{if } x^* < 0 \\ |y^*| & \text{if } 0 \leq x^* \leq x_2{}^* \\ d(p, e_2) & \text{if } x^* > x_2{}^*. \end{cases}$$

NotClose$(p, \mathcal{L})$
begin
1. $\mathcal{M} = \mathcal{L}$
2. **while** $\mathcal{M} \neq \emptyset$ **do**
3. $\quad L = \text{LastSegment}(\mathcal{M})$
4. $\quad$ **if** $d(p, L) < \delta$
5. $\qquad$ **return (false)**
6. $\quad \mathcal{M} = \mathcal{M} - \{L\}$
7. **return (true)**
end

Figure 54. Algorithm for closeness test.

A threshold value δ is assumed for the closeness test. If $d(p_0, L) < \delta$ for some previously detected line segment L, we consider p_0 belongs to this segment L and the edge tracking is suppressed even though p_0 is significant. If $d(p_0, L) > \delta$ for all previously detected L, p_0 is considered a new seed and a new edge tracking process is executed. The closeness test algorithm is shown in Figure 54.

C. RESULTS

The new method for edge detection using direction-controlled edge tracking and random hitting, described earlier with its supporting functions, was implemented on a Silicon Graphics (TM) workstation and on the autonomous mobile robot system Yamabico-11 at the Naval Postgraduate School. This section presents some of the experimental results. First, start with an image of a square object as shown in Figure 55. The processed (examined) pixels, including the 608 randomly hit pixels, are shown in Figure 56. Figure 57 shows that all four edges of the square are detected by these 608 hits. This number is less than the total number of pixels in a single row of this image (646 pixels).

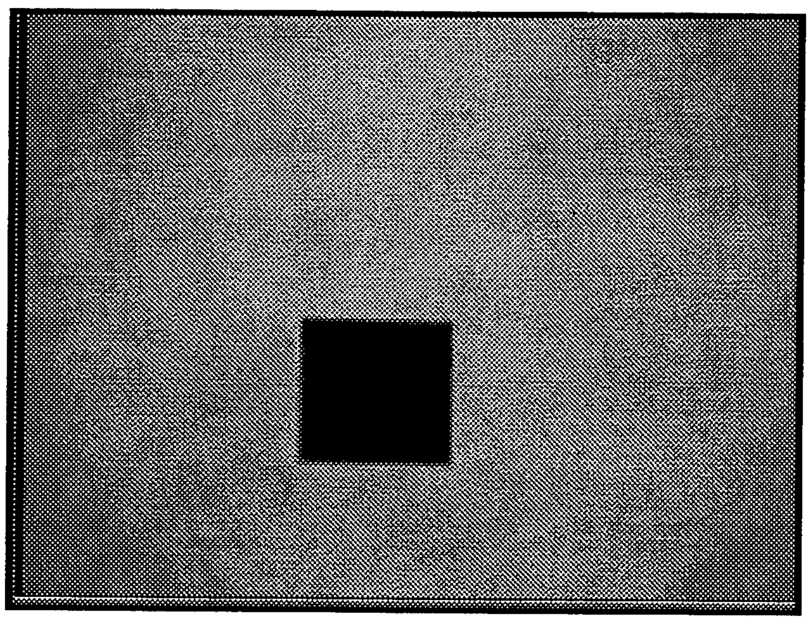


Figure 55. A square object test image.

For a sample indoor gray-scale image shown in Figure 36, we present the results of applying the algorithm. Figures 58, 59, 60, and 61 show the first 5, 10, 15, and 20 segments detected by this algorithm. In these experiments, line segments having less than 50 pixels were discarded. The total numbers of random hits needed to obtain these segments are 77, 345, 985, and 2,960, respectively. Notice that these numbers are only tiny fractions of the total image size (313,956 pixels). Thus, by a relatively

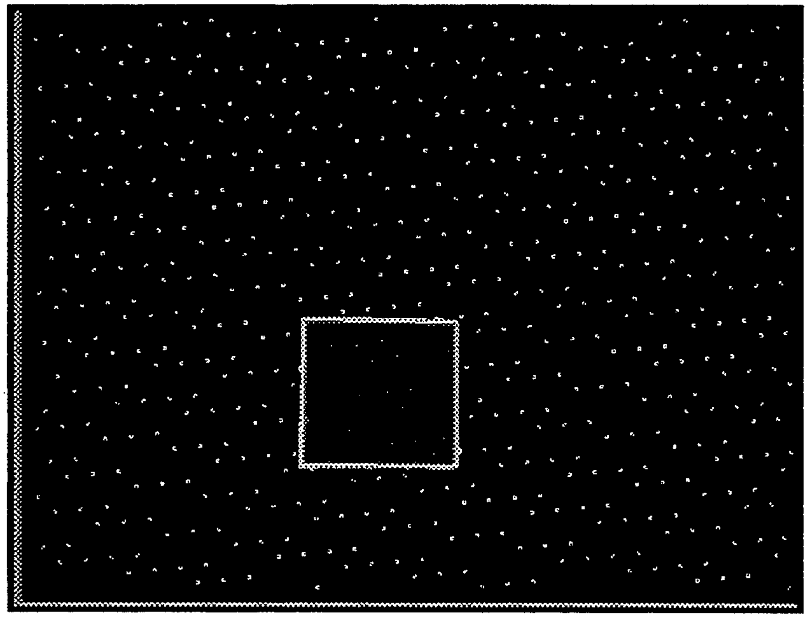


Figure 56. Processed pixels before detecting all four line segments.

small number of random hits, most of the major edges can be detected, as expected by the theoretical analysis.

Table II gives more detailed numerical data of the twenty segments detected.

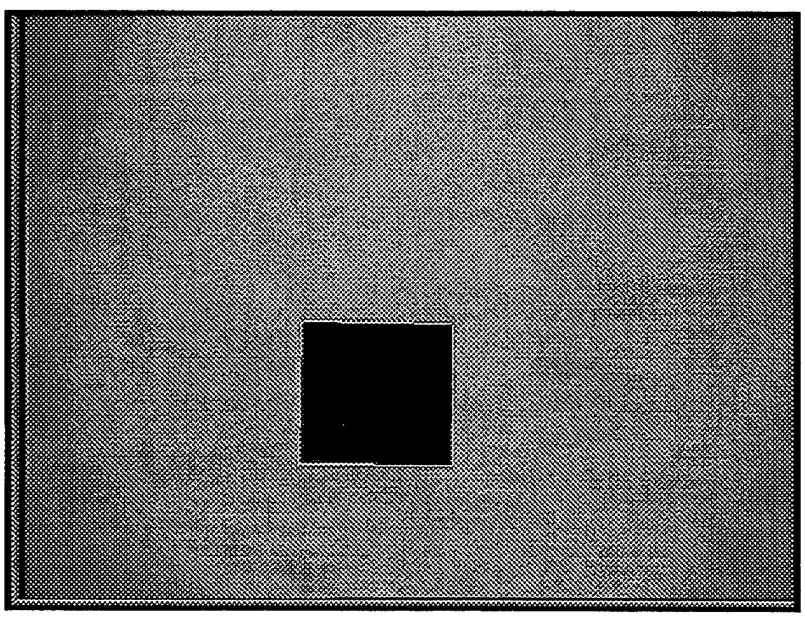


Figure 57. All four segments are detected after 608 hits.

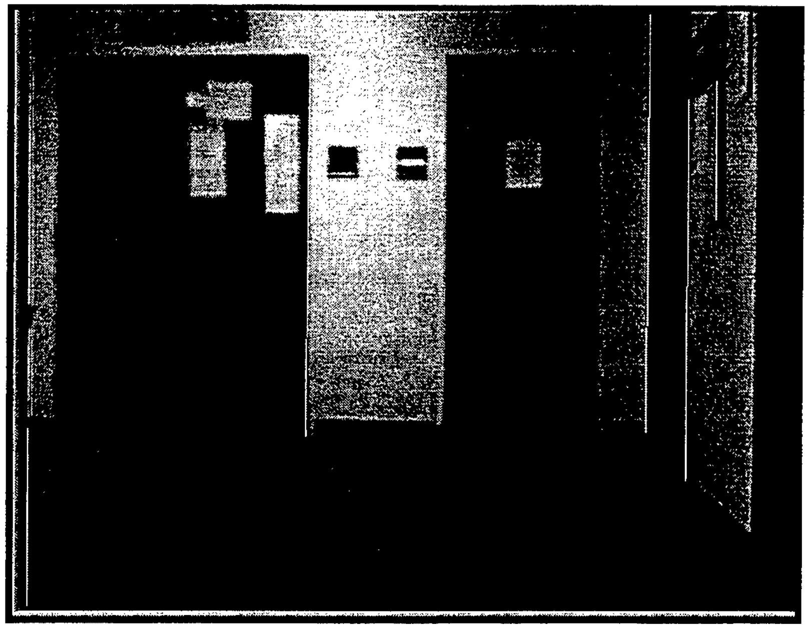


Figure 58. Five line segments detected by 77 hits.

Each row contains the data related to the ith line segment detected. For each i, H_i denotes the cumulative number of random hits done until we detect that line segment, n_i the number of pixels tracked in the segment, e_{1i} and e_{2i} its two endpoints, ℓ_i the

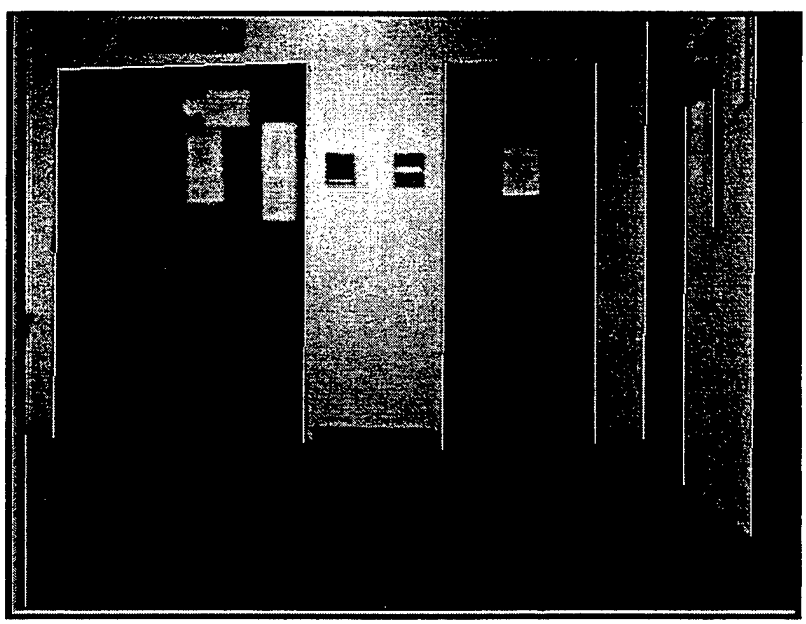


Figure 59. Ten line segments detected by 345 hits.

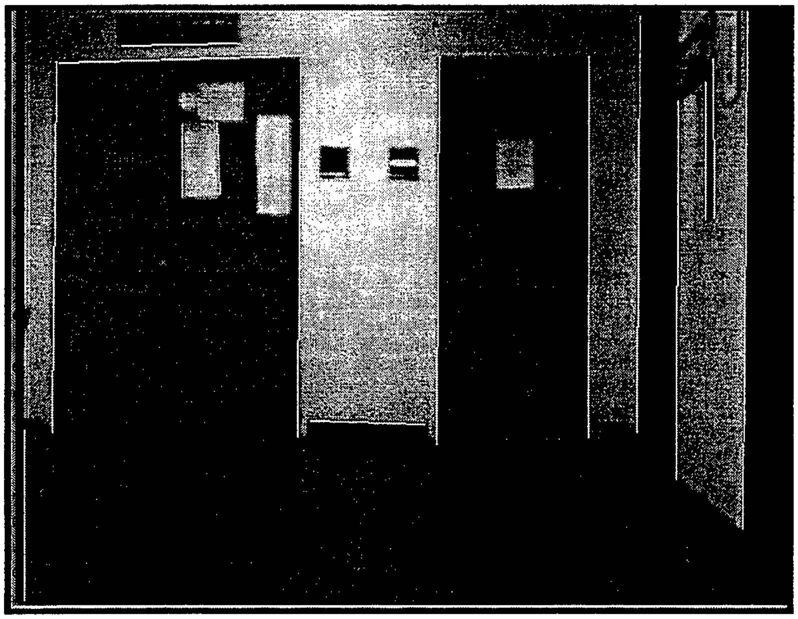


Figure 60. Fifteen line segments detected by 985 hits.

Euclidian distance between the two endpoints, α_i the orientation of the normal to the segment in degrees, and N_i the cumulative number of pixels examined in the whole process of detecting the first i line segments which also includes the number

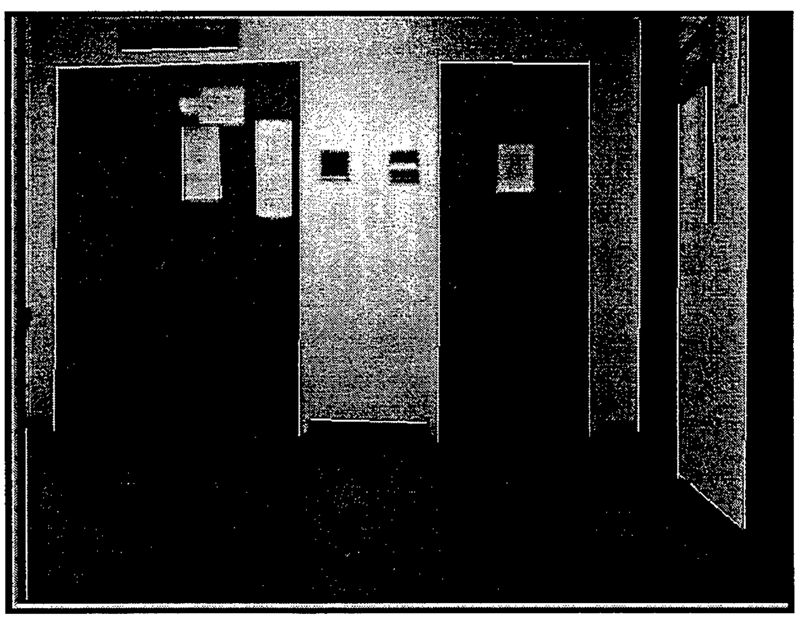


Figure 61. Twenty line segments detected by 2960 hits.

of examined pixels in segments which are not major. It was possible to detect five

i	H_i	n_i	ℓ_i	e_{1i}	e_{2i}	α_i	N_i
1	9	340	339.014	(517.308 , 138.994)	(518.762 , 478.005)	-0.246	1020
2	36	309	307.996	(236.693 , 444.002)	(235.119 , 136.010)	-0.293	1953
3	45	113	112.001	(573.715 , 312.000)	(573.666 , 424.001)	0.025	2382
4	49	141	140.000	(5.899 , 142.000)	(6.002 , 2.000)	0.042	2820
5	77	309	308.001	(551.092 , 409.009)	(549.639 , 101.011)	-0.270	3951
6	98	311	310.000	(476.939 , 444.999)	(477.042 , 134.999)	0.019	4905
7	158	316	314.993	(350.801 , 130.001)	(352.952 , 444.987)	-0.391	5865
8	203	203	202.114	(33.941 , 440.269)	(235.986 , 445.542)	-88.505	6543
9	249	302	301.015	(29.333 , 139.997)	(31.992 , 441.000)	-0.506	7527
10	345	426	424.004	(605.151 , 59.999)	(608.068 , 483.993)	-0.394	8820
11	503	61	60.008	(136.598 , 389.994)	(137.435 , 329.992)	0.799	9162
12	517	98	97.021	(183.013 , 456.357)	(86.013 , 454.378)	-88.832	9456
13	600	97	96.047	(245.001 , 149.018)	(340.997 , 145.910)	88.145	9840
14	973	131	130.004	(580.366 , 443.011)	(581.390 , 313.011)	0.451	10611
15	985	59	58.026	(169.100 , 333.057)	(166.084 , 391.004)	2.979	10788
16	1028	78	76.987	(198.791 , 395.978)	(200.183 , 319.003)	1.036	11022
17	1744	76	67.140	(549.703 , 104.654)	(600.681 , 60.961)	49.400	11961
18	2622	75	74.076	(598.522 , 409.938)	(600.329 , 483.992)	-1.398	12825
19	2752	79	78.000	(228.174 , 318.000)	(228.180 , 396.000)	-0.004	13062
20	2960	125	124.008	(351.995 , 443.870)	(476.001 , 443.330)	89.750	13716

Table II. Data of line segments obtained by the tracking algorithm.

line segments by examining 3951 ($=N_5$) pixels, which is only 1.25% of the total pixels (313,956). Likewise, to detect 10, 15, and 20 line segments, 8820, 10788, and 13716 pixels had to be processed, which are equal to only 2.8%, 3.43%, and 4.36% of the total pixels, respectively.

The graph of H_i as a function of i is shown in Figure 62. The time to detect the $(i+1)^{\text{st}}$ line segment is generally larger than the time for the i^{th} line, because the probability of hitting "undetected" segments becomes lower.

Using several indoor scenes, we made experiments to find the distribution of the lengths of the edges detected from typical indoor images. We limit the maximum number of hits to only 3000 for these images. The following table shows the results for five test images taken in different positions in the hallway and in the lab. Each

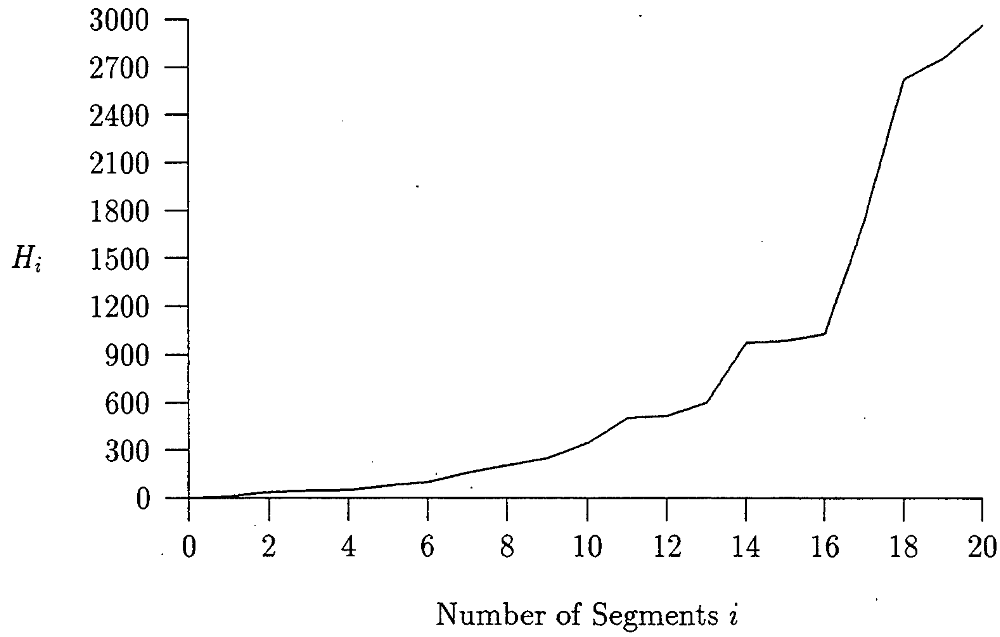


Figure 62. Number of random hits H_i to obtain first i segments.

entry in the table shows the total number of obtained segments, and the number of lines in each of the three ranges of the lengths ℓ (in pixels) of these segments for each image. The results show that the majority of the line segments in the images have

Image	Num. of Segments	$75.0 \leq \ell < 100.0$	$100.0 \leq \ell < 300.0$	$\ell \geq 300.0$
1	21	2	17	2
2	21	6	12	3
3	23	6	15	2
4	10	3	6	1
5	26	6	17	3

Table III. Number of line segments from several images.

lengths in the range of 100-300 pixels, within which most of the landmarks in the indoor images fall.

We also made experiments of detecting only vertical edges with a maximum of 3000 hits, since vertical edges are essential in some of the image understanding tasks in robotics. The results obtained from three images sequentially taken in the hallway are shown in Table IV. Each entry in the table contains the number of vertical line segments detected and how many hits ($H < 3000$) were actually needed for each image. The segments obtained are shown in Figures 63, 64, and 65.

Image	Num. of V. Lines	Num. of Hits
1	13	2810
2	12	2857
3	15	2553

Table IV. Number of vertical line segments from a sequence of three images.

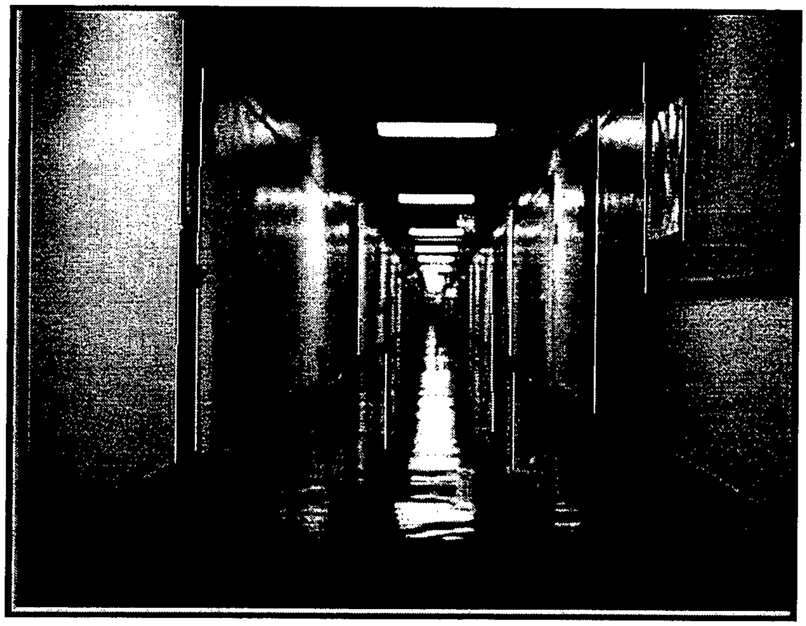


Figure 63. Thirteen vertical edges detected by 2810 hits.

Some line segments are detected due to reflections on the floor. These lines can be interpreted if we have a world model, the camera pose is known, and a proper pattern matching is performed.

D. SUMMARY

In this chapter, we have presented a global algorithm for a new efficient edge detection method that avoids exhaustive image scanning. We use a pseudo-random number scheme to find a starting pixel from which edge tracking for a linear sequence of pixels is performed. The directional information of the line segment is used to

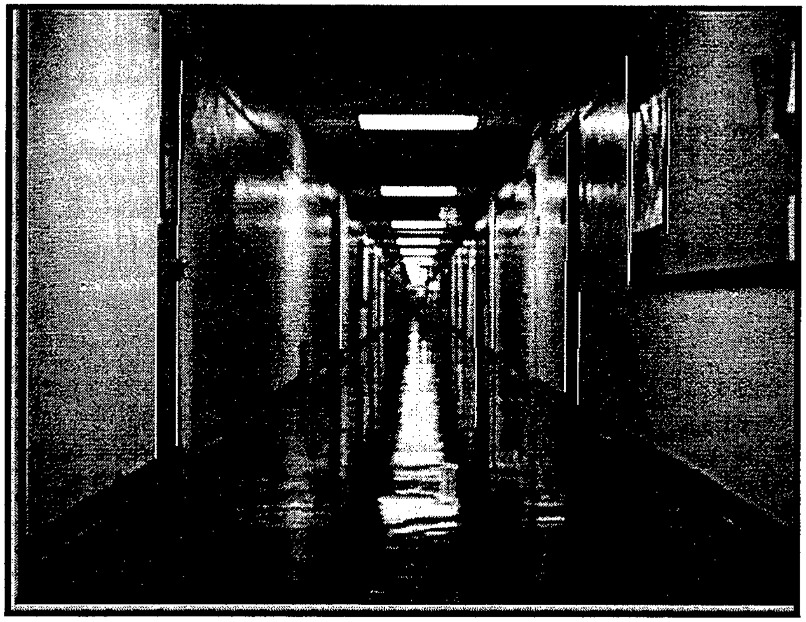


Figure 64. Twelve vertical edges detected by 2857 hits.

control the edge tracking. A robust least-squares fitting method is used to obtain the geometric features for the line segment that best fits the pixel sequence, including its endpoints. During this new edge detection method, redundant tracking of an edge

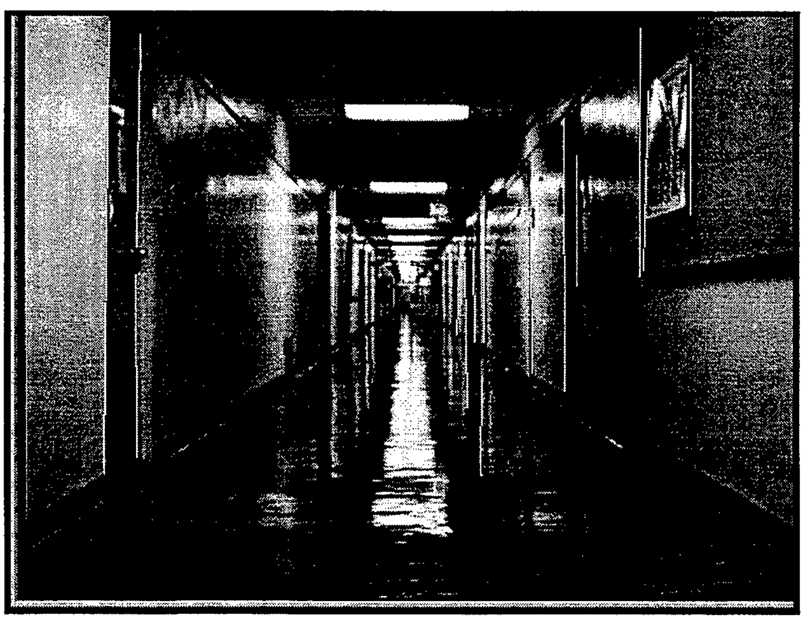


Figure 65. Fifteen vertical edges detected by 2553 hits.

is avoided by the closeness test. In real experiments, we have been able to detect most of the major edges in a test image of size 313,956 with a very small number of pixels processed (4.36% of total pixels). In other experiments on some indoor images, we detected more than twenty major edges, in the range of 75 pixels to more than 400 pixel lengths, by less than 3000 hits. The algorithm also has been tested to detect vertical edges only. Most of the vertical edges in indoor images were detected by a total number of hits which is less than 1% of the total number of pixels in each image. The results obtained from these experiments show that this algorithm is efficient and useful for numerous image understanding applications and autonomous robot navigation.

VI. WORLD MODEL

The geometrical model of the robot's environment is essential for the interpretation of detected edges which can be used in the self-localization task. Also, the model is important for detecting obstacles, motion planning, and many other tasks for the robot.

A. BACKGROUND

Our efforts toward developing image understanding algorithms are aimed to be used by the autonomous mobile robots for model-based visual navigation. For the autonomous mobile robot Yamabico-11 at the Naval Postgraduate School, the current operating environment is part of the second floor in the Spanagel Hall building. In this chapter, we consider the problem of modeling that environment. Some of the capabilities needed for the model of the world are based on previous work done on the fifth floor of the same building [Ref. 36], as well as some techniques of computer graphics for perspective projection [Ref. 73]. By examining the old model, we noticed that some simplifications are needed for real-time navigation. We expect the following three major features for the new model:

1. Simplicity
2. Flexibility
3. Accuracy

A simple model structure has an advantage of saving computation time over a complex one. By flexibility, we mean that a user can easily modify some portions of the model, for instance, to identify additional obstacles in the world without changing the modeling algorithm itself. By accuracy, we mean that the model should represent the main landmarks of the environment precisely. This feature is important to support self-localization (pose determination), obstacle avoidance, and path planning with minimal errors; the model must reflect exactly what the robot expects to "see" from its position and orientation.

B. 2D POLYGONAL WORLD

1. Polygons

a. *General Definitions*

Polygons are used as the basic building blocks for the world representation. A polygon $\mathcal{B}$ is represented by an ordered set of n distinct vertices $\{v_1, v_2, \cdots, v_n\}$, such that $n \geq 3$. Some examples of polygons are given in Figure 66. A basic function φ used in representing a polygon is one that determines the *next*

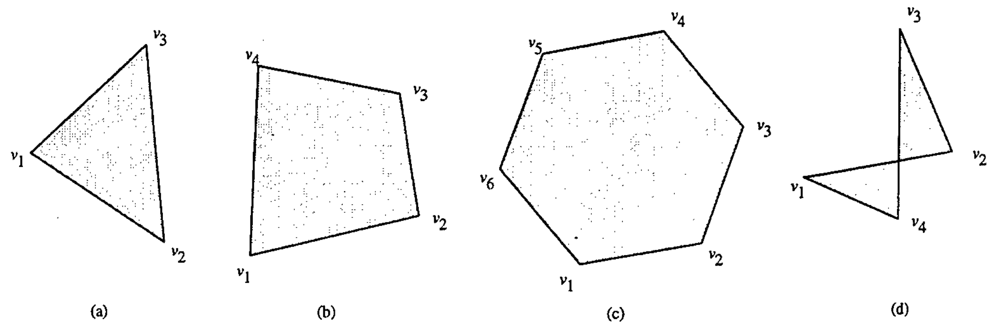


Figure 66. Examples of polygons.

vertex for each vertex v_i. That function $\varphi : \mathcal{B} \rightarrow \mathcal{B}$ is defined as follows [Ref. 68]:

$$\varphi(v_i) = \begin{cases} v_{i+1}, & \text{if } 1 \leq i < n. \\ v_1, & \text{if } i = n. \end{cases}$$

An example of such a polygon's representation is shown in Figure 67. The polygon shown has four vertices v_1, v_2, v_3, and v_4 where $\varphi(v_1) = v_2$, $\varphi(v_2) = v_3$, $\varphi(v_3) = v_4$ and $\varphi(v_4) = v_1$. The *next* function φ is a bijection, and hence, there exists an inverse function $\varphi^{-1} : \mathcal{B} \rightarrow \mathcal{B}$ such that $\varphi^{-1}(v)$ determines the "previous vertex" of v. A segment $\overline{v\varphi(v)}$ is called an *edge* joining vertex v to its next vertex $\varphi(v)$.

Definition: A polygon $\mathcal{B}$ is called *simple* if it satisfies the following two conditions:

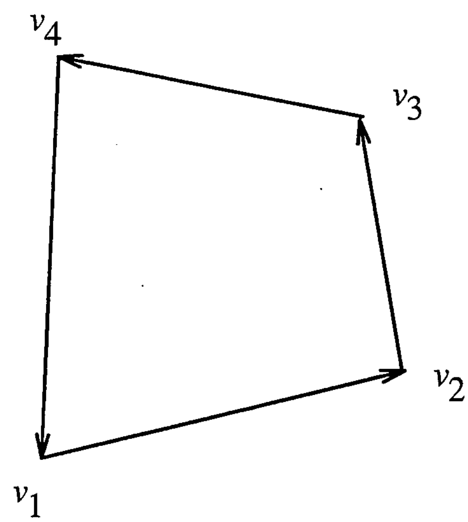


Figure 67. A polygon.

1. No triple of vertices $(v, \varphi(v), \varphi^2(v))$ in B are collinear,
2. There is no pair of non-consecutive edges sharing a point in $\mathcal{B}$ [Ref. 68].

In Figure 66, there are three simple polygons (a,b, and c) and one non-simple polygon (d).

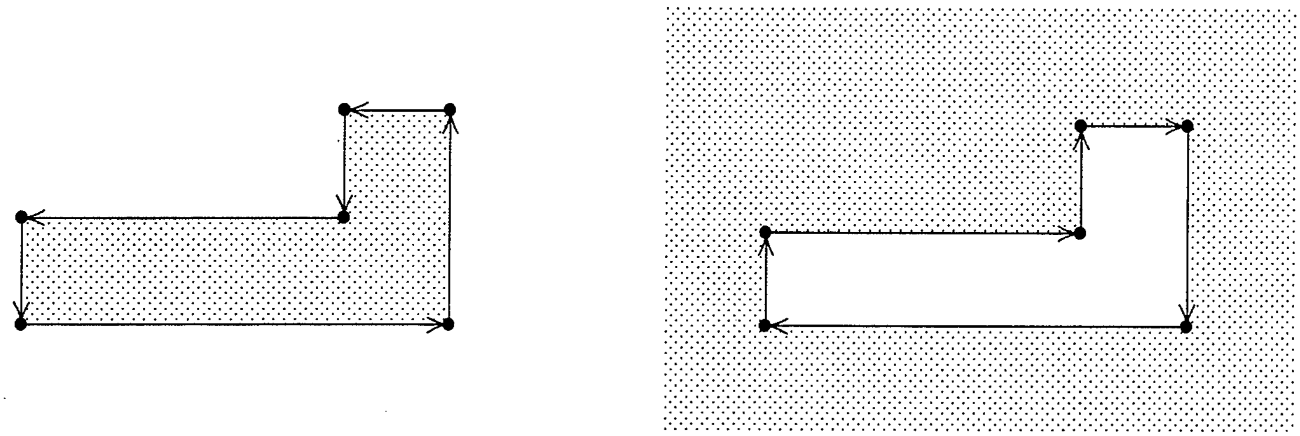
In our environmental modeling, only simple polygons are used. There are two types of polygon, based on the direction in which its vertices are traversed (Figure 68).

- Counterclockwise (CCW) polygon, which is used to represent an obstacle in the operating environment.
- Clockwise (CW) polygon, which is used to represent an outermost boundary of the operating environment.

b. Data Structures

A polygon is represented as a doubly-linked list of vertices. The representation of each vertex v_i has its own (x, y) coordinates, a pointer to its *next* vertex $\varphi(v_i)$, and a pointer to its *previous* vertex $\varphi^{-1}(v_i)$ (Figure 69).

In Figure 70, we show a data structure for the polygon shown in Figure 67.



(A) CCW polygon (B) CW polygon

Figure 68. CW and CCW Polygon.

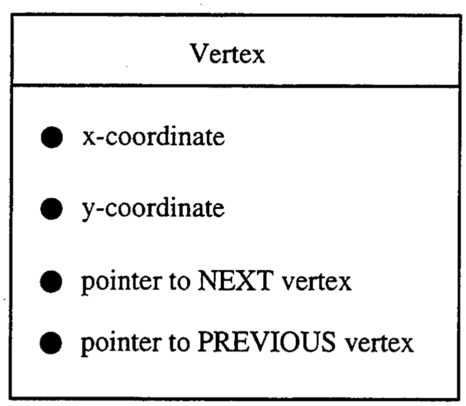


Figure 69. Data structure of a vertex in a polygon.

2. World

A vehicle's 2D environment is represented by a polygonal world model. Generally, a world $\mathcal{W}$ is bounded by an outermost CW polygon and contains zero or more

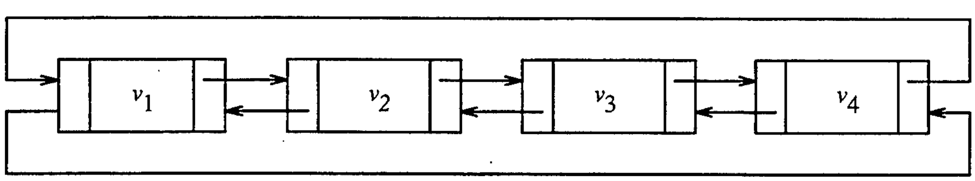


Figure 70. Data structure of a 4-vertex polygon.

of CCW polygons (obstacles) inside that boundary. An example of a polygonal world is given in Figure 71.

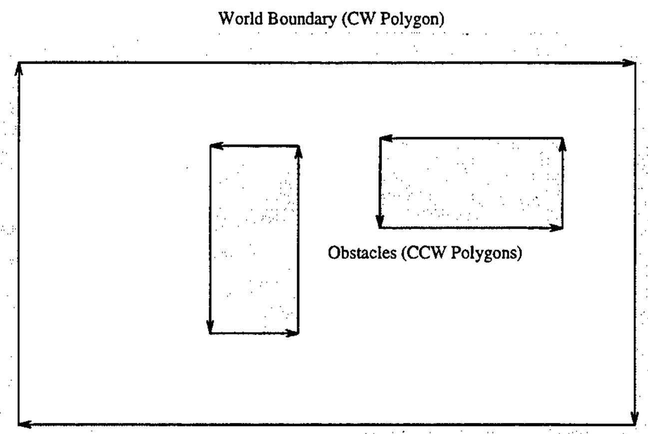


Figure 71. A simple polygonal world.

a. *World Data Structures*

A world model consists of a linked list of polygons. As described previously, each polygon is represented by a doubly linked list of its vertices. Figure 72 illustrates the general data structure used for the representation of a 2D world model.

b. *Yamabico's 2D World*

For our vehicle's environment, we choose to represent the 2D world with one CW polygon to represent the boundary of the floor. CCW polygons (obstacles) were not included in this stage of implementation. However, they can be added later to the model, if it is needed. The world coordinates are shown in Figure 73. Orientation of a camera or a vehicle with respect to the model is shown also. Each point on the floor where a vertical edge stands is described by its position (x, y) and pointers to its next and previous vertices. Figure 74 shows the actual 2D representation of the floor of the environment of the robot *Yamabico*.

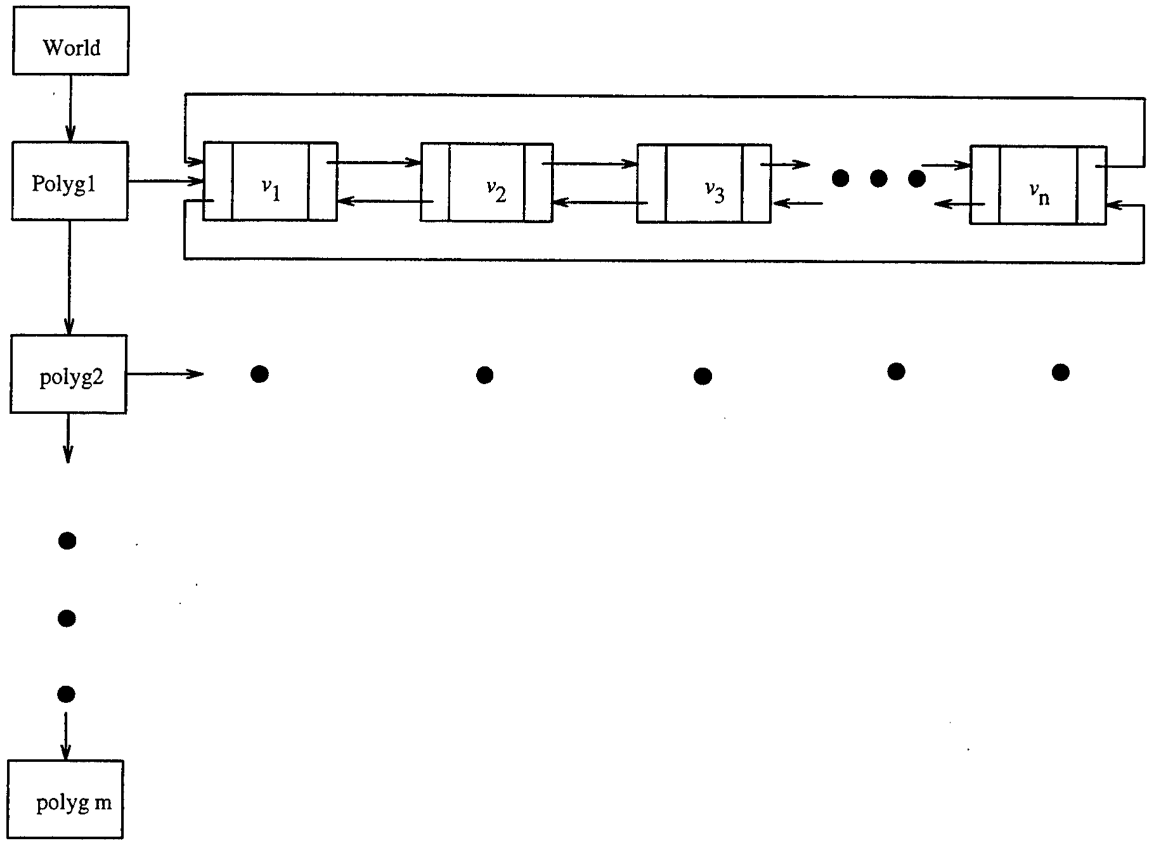


Figure 72. Representation of 2D world data structure.

C. EXTENDING 2D MODEL TO 3D

Although the 2D representation is appropriate for motion planning, it is not enough for the image understanding that requires additional information from the environment. For example, some obstacles cannot be detected by certain sensors, such as sonars. The motivation of using a camera and a vision system is to provide information about those objects in the world that are difficult to detect with sonars. Also, sensing the walls and other 3D objects requires the addition of *height* information to the current 2D model.

Due to the consideration of doors and walls as landmarks for visual navigation tasks, it is an important issue to represent their locations with respect to the 2D model. For that reason, they have to be modeled in a way that gives the interpretation of these landmarks when the vehicle performs its motion planning. Figure 75 shows

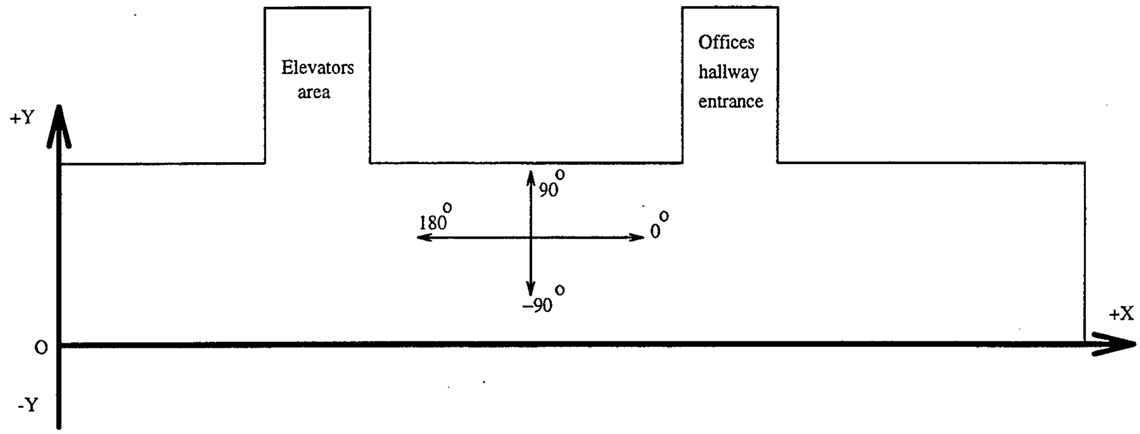


Figure 73. World coordinates and orientation of 2nd floor model.

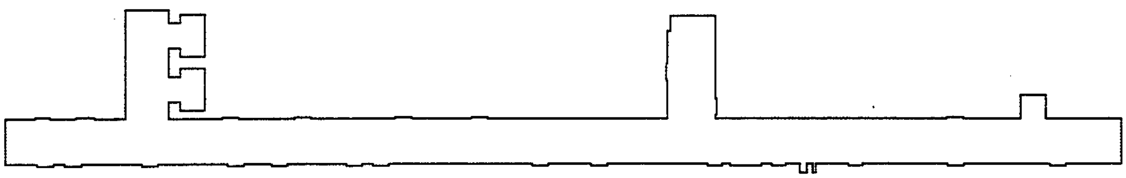


Figure 74. 2D representation of the hallway.

a simple 3D world consisting of a floor, ceiling, walls, door, and door ceiling.

The environmental 3D model of the second floor in Spanagel Hall has one floor, three main ceilings, and several door frame "ceilings". We represent each of those as a polygon. To obtain 3D modeling, a *height* value should be associated with each polygon in the model. Thus, the height value of the floor polygon equals zero, and for a ceiling polygon it takes the actual height of the ceiling measured from the floor. We call these polygons representing floor or ceilings *world components*. Connecting the vertices from the different world components provides the 3D model. For example, connecting those vertices on the floor to their corresponding vertices on the ceiling of a door frame by vertical edges gives the information on the door frame. A simple example is shown in Figure 76. In this figure, the door frame is described by a sub-polygon {B,C,D,E}, a polygon {G,H,I,J}, and vertical edges $\overline{BG}, \overline{CH}, \overline{DI}$, and $\overline{EJ}$. Additional description about 3D model are given in [Ref. 36].

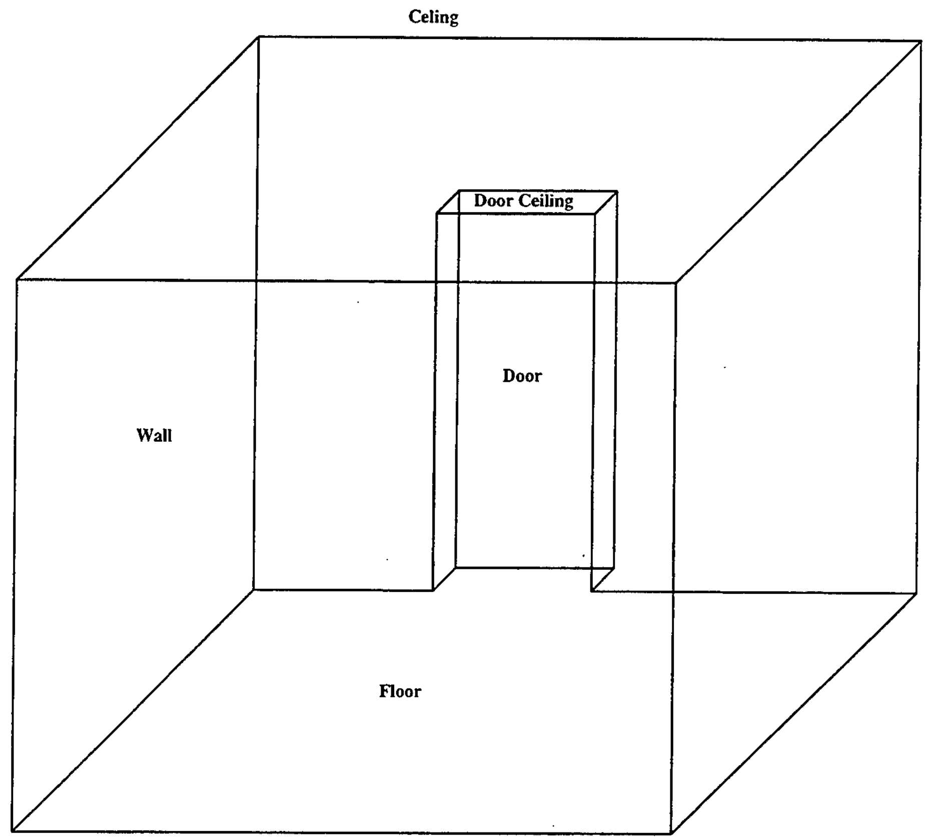


Figure 75. A simple 3D world.

D. $2\frac{1}{2}$D MODEL

A complete 3D model may well be suitable for computer graphics applications. However, for robot navigation, representing detailed descriptions of environments such as ceilings may practically not be needed. Thus, we propose simplifying the model of the environment to gain more *efficiency* when it is interpreted by some visual tasks of the robot. We consider the world as a floor, *plus* walls and door frames only. The walls and doors are represented as vertical edges *extending* from the floor. We call that model a **$2\frac{1}{2}$D** model. We expect that other portions of the world are not essential. Figure 77 shows the $2\frac{1}{2}$D representation of the 3D world shown in Figure 75.

That model is mainly the 2D representation of the floor, *plus* the information of the height of all model vertical lines extending from the floor. A world view should show only the vertical model edges. This is intended to save the computational time

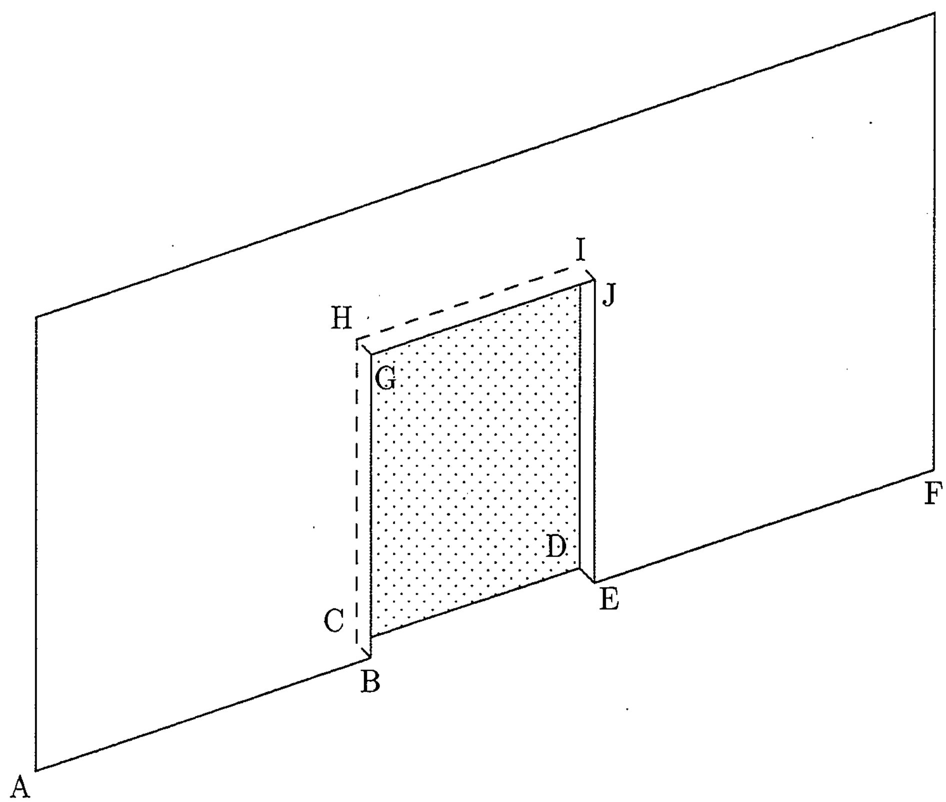


Figure 76. Representation of a door frame in the hallway model.

required to build a complete 3D model which includes details not relevant to the visual navigation of a robot. For example, the ceilings are higher than the vehicle and representing them is a waste of storage and increases the processing time. In Figure 78, the description of the data structure for a vertex in this model is given.

The new model is obtained by:

1. Eliminating any additional representation in the model data structures other than the 2D polygonal world of the floor, such as ceilings.

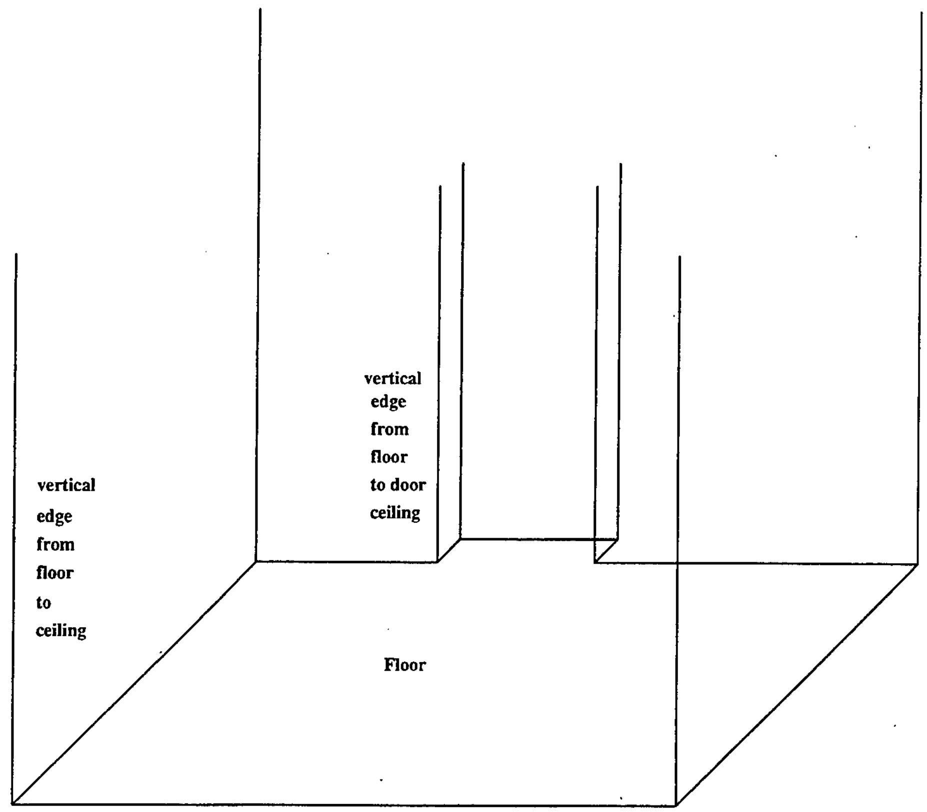


Figure 77. A simple $2\frac{1}{2}$D world.

2. Adding the height (z-value) of any vertical edges extending from each vertex in the floor.

3. Eliminating complex functions that deal with 3D geometrical structures (such as constructing ceilings and connecting their points to their correspondent points in the floor by vertical edges).

4. Computing model vertical lines directly from the data structures of the floor vertices.

In APPENDIX: WORLD MODEL, the model description of the current operating environment of the autonomous robot Yamabico-11 is given.

E. TRANSFORMING MODEL INTO 2D VIEW

The robot must find a 2D scene from a given viewpoint p_0 and the world model. After defining the data structures of the model, some graphics functions for

Vertex
• x-coordinate • y-coordinate • z-value of its vertical edge • pointer to NEXT vertex • pointer to PREVIOUS vertex

Figure 78. Data structure of a vertex in a polygon within the $2\frac{1}{2}$D model.

transforming the model edges into 2D view line segments can be used to perform that task [Ref. 36, 73]. This task includes the following steps:

- Finding the visible points of the hallway floor from p_0 for the 360° angle, regardless of its orientation.
- Finding the vertical edges extending from the visible floor points obtained in the previous step.
- Specifying the 2D viewing parameters.
- Projecting extracted features (lines) into the 2D viewing window.

The problem of visibility is to determine a set of visible points in the world from the given position of the viewing point. We call this process *point visibility.* To illustrate this concept, a simple example of a polygonal world is given in Figure 79, which shows the visible points from the viewing point p_0 marked by a black circle at each. Several methods for visibility testing can be used. In our implementation of the environmental modeling for Yamabico, some of the vision system library (on an SGI machine) included in [Ref. 36] are portions used for the visibility testing.

The *perspective projection* is a well-known technique in computer graphics and computer vision. The viewing volume for a perspective projection is called a *frustum,* or a truncated pyramid [Ref. 74]. This is shown in Figure 80. Usually the window

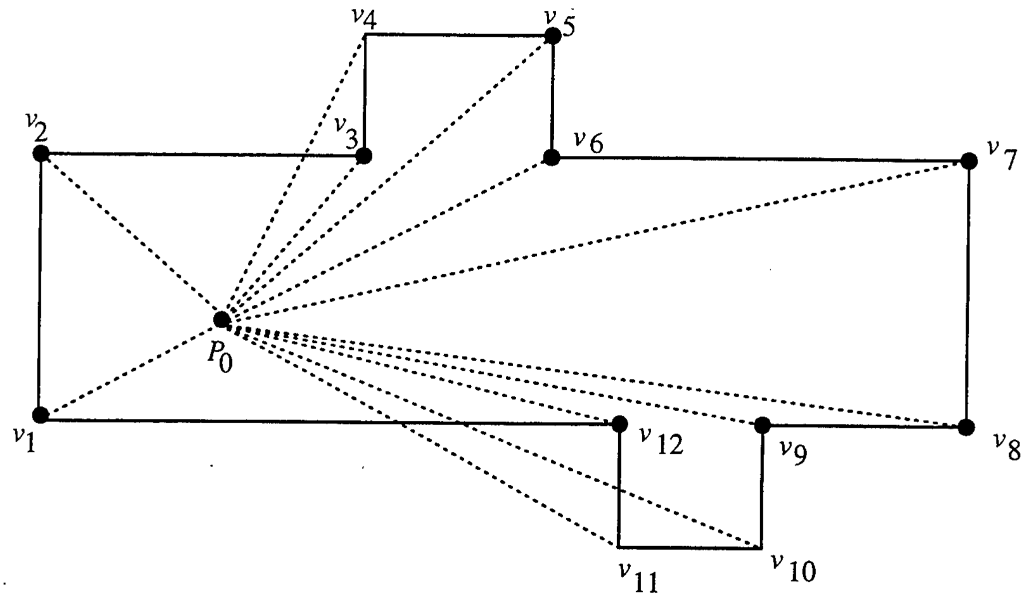


Figure 79. Example of point visibility.

width and height are the same as the dimension of the image produced by the camera of the vision system. Values of near clipping and far clipping planes should be chosen appropriately to give the viewer a good sense of the spatial relationship between different parts of the world [Ref. 73]. One strategy in the modeling is to set the near clipping value to the focal length of the camera's lens and the far clipping to be larger than the distance between the origin of the world and the farthest point in the world from the origin. The near clipping plane is used to clip objects too close or behind the viewer (camera), while the far plane clips those objects too far to be seen [Ref. 75].

The projection from model lines to 2D view includes several steps, as shown in Figure 81 and described in [Ref. 73]. They are provided for the vision system library and implemented by [Ref. 36].

However, in the case of the on-board image understanding algorithms for the vehicle, the result is not intended for display. It will be stored in data structures defining the information of the virtual line segments that are expected to be viewed. The information for each line segment should include its two endpoints within the

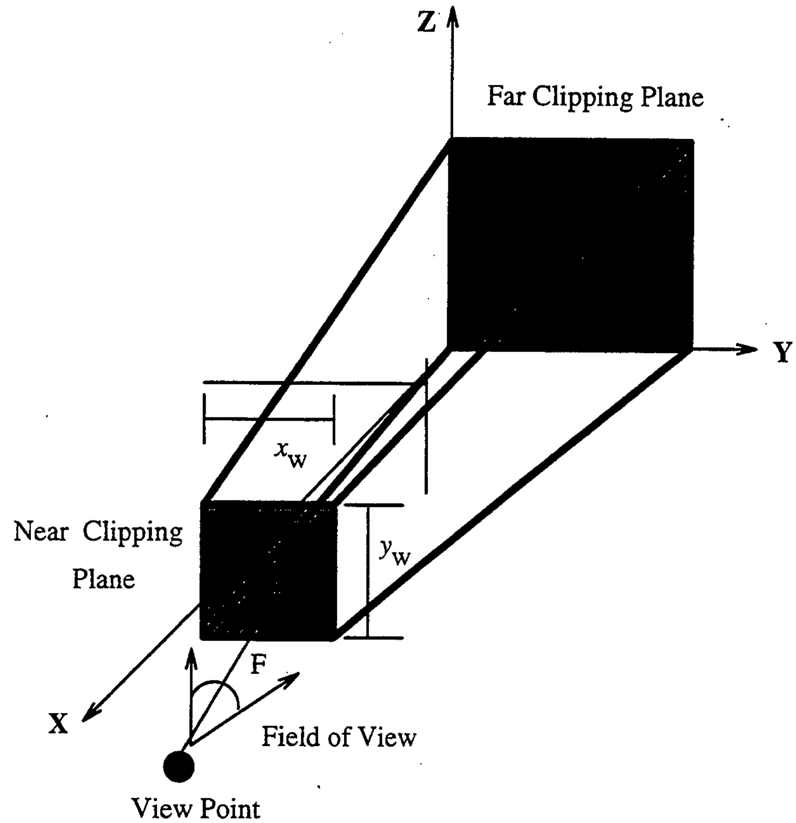


Figure 80. Truncated perspective viewing volume.

view plane. This will be used for the matching procedures against the actual line segments detected from images taken by the vehicle's camera.

F. MODELING RESULTS

To implement the model of the second floor on Yamabico and integrate this implementation with the current vision system and MML, it is necessary to test the model visually through a graphical workstation to check whether it correctly reflects

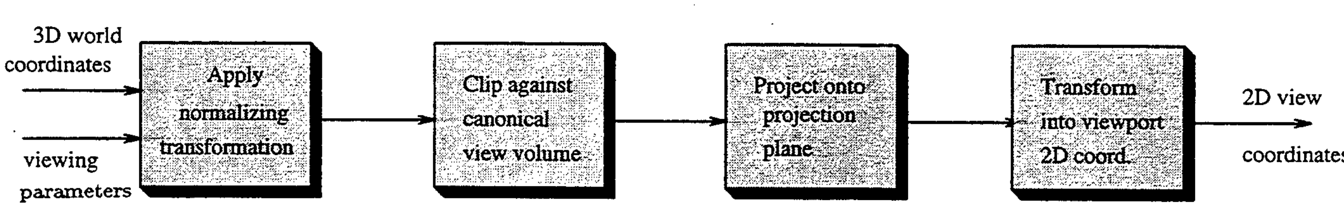


Figure 81. Steps of 3D world coordinates to 2D view projection.

the basic features of the environment. In this implementation, both the 3D model and the $2\frac{1}{2}$D model were tested from several estimated positions.

Figure 82 shows one 3D model view from a position of the camera estimated by x=2032 cm, y=124.25 cm, with the viewing orientation of 0 degree (as if the camera is positioned in the front of the AI and Robotics lab). The field of view angle was set to 64°, while the focal length was 3.0 cm. The view of the $2\frac{1}{2}$D model from the same

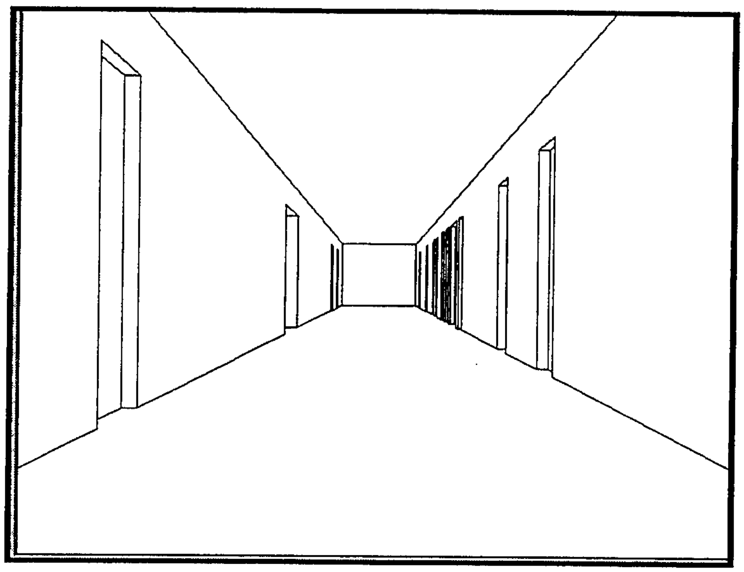


Figure 82. Hallway view from x=2032 cm, y=124.25 cm, $\theta = 0°$, view angle=64° and focal length=3.0 cm.

position using the same focal length and field of view angle is shown in Figure 83.

By changing the value of x to 3000 cm, as if the camera were moved 968 cm from its previous position, the view of the 3D model shown in Figure 84 was obtained. The corresponding view of the $2\frac{1}{2}$D view is shown in Figure 85.

This method was efficient enough. Through these experiments, we concluded that the model can be added easily for the autonomous mobile robot Yamabico's motion planning tasks. One interesting application for which we want to use that model, together with the new efficient edge detection method, is to obtain a fast vision-based pose determination algorithm, as we will describe in the next chapter.

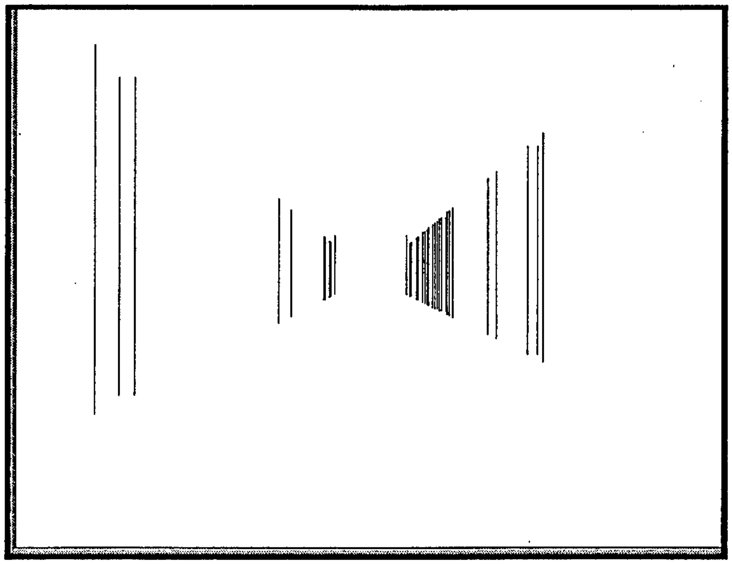


Figure 83. Vertical edges of the hallway using $2\frac{1}{2}$D model from the same view point as in the previous figure.

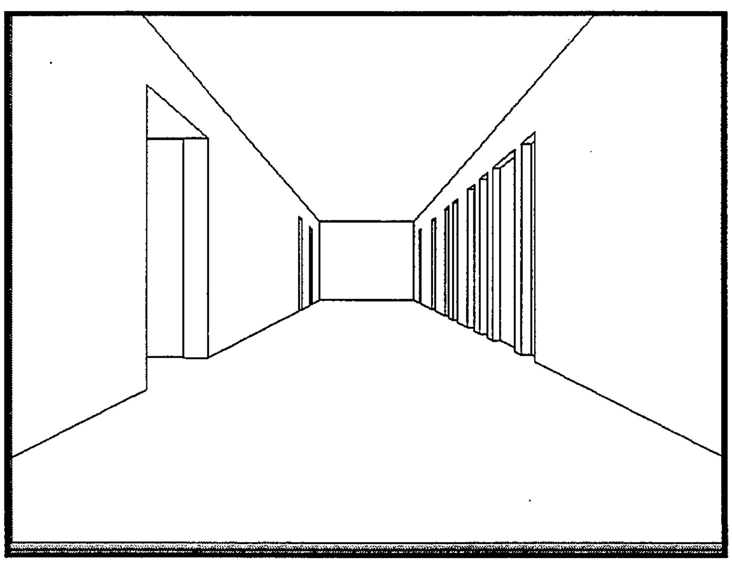


Figure 84. Hallway view of the 3D model from x=3000 cm, y =124.25 cm, $\theta = 0°$.

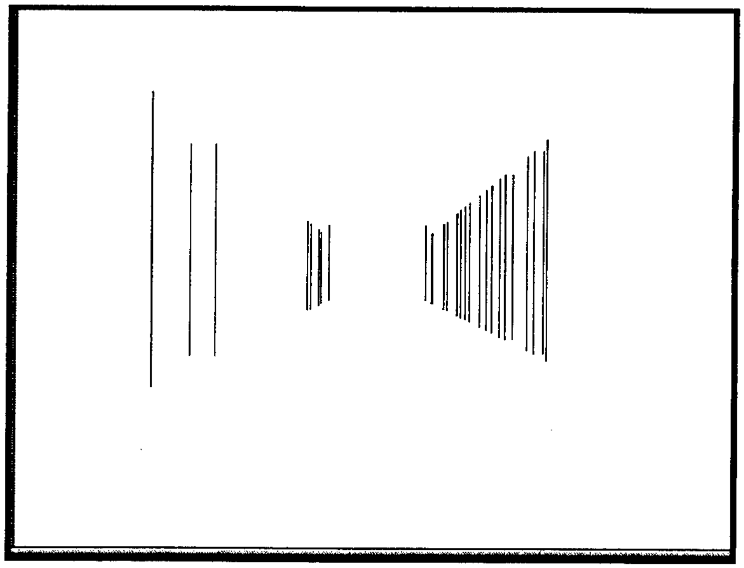


Figure 85. Vertical edges of the hallway using $2\frac{1}{2}$D model from the same view point as in the previous figure.

VII. APPLICATION OF EDGE TRACKING AND MODELING: VISION-BASED POSE DETERMINATION

A. INTRODUCTION

In the previous chapter, the modeling of the environment in which an autonomous vehicle navigates was described. One of the tasks to be addressed with such a model is the correction of the position and orientation of the vehicle with respect to the model. This is one of the important problems for mobile robots. A common method of positional identification for an autonomous vehicle is *odometry*, or *dead reckoning*. However, this capability is affected by accumulated errors due to wheel slippage and an uneven floor. Due to these effects, uncertain estimates of the position and orientation may affect the vehicle's motion. For the autonomous mobile robot Yamabico, localization using ultrasonic sensors, 2D transformation, and linear fitting of the sonar return points [Ref. 11] was successfully performed. Some work has been done on a graphics workstation using a complete 3D model of the previous environment of the vehicle (fifth floor of Spanagel Hall) and an exhaustive-scanning edge detection method [Ref. 35]. However, by making experiments based on that work, we found that it takes 19-20 seconds to correct the estimated pose. Thus, we decided to solve the problem in an improved and more efficient way so that it will be suitable for further visual navigation tasks of the robot in its current environment.

Our work is mainly aimed at efficiency and robustness improvement in practical image understanding. Through our proposed improved solution to the problem, the work mainly includes the following two significant approaches:

1. Applying our new edge detection algorithm using direction-controlled edge tracking and random hitting [Ref. 70], as described in Chapters IV and V.
2. Using the proposed $2\frac{1}{2}$D model rather than using a complete 3D model.

Our geometrical solution to the problem is presented in this chapter as well.

B. PROBLEM STATEMENT

The problem of pose determination and correction for an autonomous robot's visual navigation in an indoor environment can be stated as follows: Given an estimated pose $q_e = (x_e, y_e, z_e, \rho_e, \varphi_e, \theta_e)$ where ρ_e, φ_e, and θ_e are the roll, pitch and yaw angles of a camera, an input image $\mathcal{I}$ taken by the camera, a description of the environment (world) model M, and the focal length f of the camera, compute the correct pose q of the camera (Figure 86). We assume that if the optical axis of the camera is parallel to the floor($\rho_e = 0$ and φ_e=0), then the vertical edges in the real world will appear vertical in the image. We call the positions in the 2D world where the vertical edges stand *control features.*

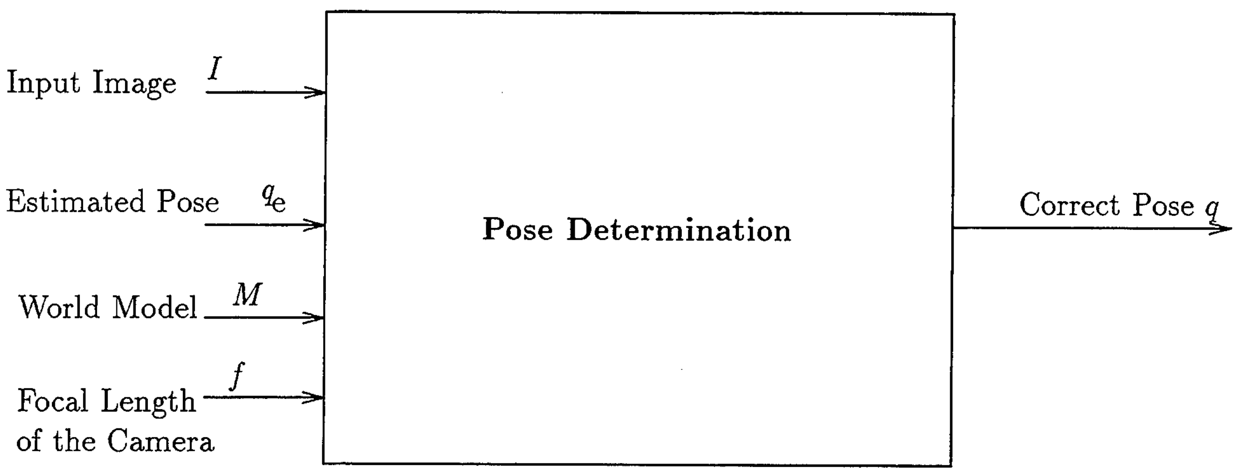


Figure 86. Pose determination problem

C. OVERVIEW

First, we adopt the solution of the problem of point location using distinguishable marks, proposed by K. Sugihara ([Ref. 64]). The solution idea is used as a basis for the overall algorithm of the pose determination and correction.

1. Sugihara's Formulation and Solution of Point Location Problem (PLP)

Let p_1, p_2,..,p_k be points on a plane and $B = (e; r_1, , ..., r_k)$ be a bunch of rays. Find the motion M such that the ith ray in $M(B)$ passes through p_i for all

$i = 1, ..., k$.

In [Ref. 64], the rays are considered to represent the directions in which a set of k control features (vertical edges) in the world are observed. Additionally, consider a situation where these rays are passing through image plane's extracted vertical edges. The solution addressed there was that if k=3, the camera position is usually determined uniquely.

2. Camera Pose Determination as PLP

We think of the motion M as the **pose** of the camera (robot) defined by :

- The **position** v of the camera in the plane, where the rays r_1, r_2,..,r_k are originating from v to some other points (p_1, p_2,..,p_k) in the plane.
- The **orientation** θ of the camera heading.

To apply the solution idea in [Ref. 64], let $\beta_1 = \angle p_1 v p_2$ be the viewing angle from v to p_1 and p_2, and $\beta_2 = \angle p_2 v p_3$ be the viewing angle from v to p_2 and p_3. Since we have two constant angles β_1 and β_2, and there is only one possible point in the given world which has viewing angles β_1 and β_2 between the control features p_1, p_2, and p_3, then the unique position should be the precise camera position. It is determined by a point v on the intersection of two circles, one passes through p_1 and p_2 such that the angle of the arc p_1 p_2 is equal to β_1 and the other passes through p_2 and p_3 where the angle of the arc p_2 p_3 is equal to β_2 (Figure 87).

D. ALGORITHM

The algorithm is described in Figure 88. Using the $2\frac{1}{2}$D model, we find the 2D view M_1 of the $2\frac{1}{2}$D model M that represents those vertical edges that are expected to be seen by a camera with a focal length f at a given pose q_e (Line 1), as in Figure 89. A vertical line in this 2D view should correspond to a single vertical edge in the modeled world. Using our straight edge detection method, we find a set M_2 $=\{l_1, l_2, l_3\}$ of the detected vertical line segments $\mathcal{L}$ (Lines 2 and 3).

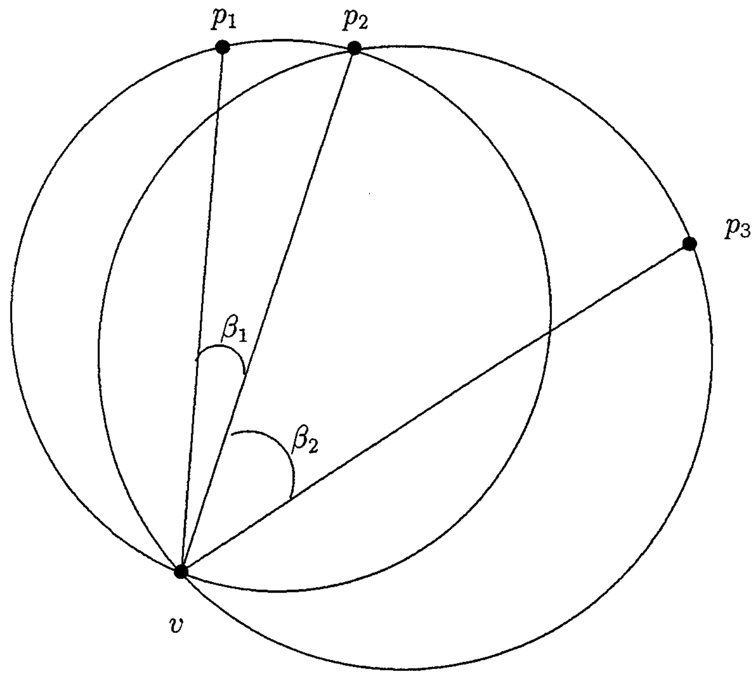


Figure 87. The intersection of two circles passing through three control features uniquely determines the camera position.

CorrectPose$(f, M, \mathcal{I}, q_e)$

begin
M_1 = ProjectModel(q_e, f, M)
$\mathcal{L}$ = detectEdges$(\mathcal{I})$
M_2 = Longest3VerticalEdges$(\mathcal{L})$
for (each $l_i \in M_2$)
    e_i = FindBestMatchedModelVerticalLines(l_i, M_1)
    p_i = 2DworldCoordinatePosition(e_i)
β_1 = ViewAngle(l_1, l_2, f)
β_2 = ViewAngle(l_2, l_3, f)
v = FindActualPose$(p_1, p_2, p_3, \beta_1, \beta_2, q_e)$
θ = CorrectOrientation$(f, v, p_1, p_2, p_3, \mathcal{I}, M_2)$
return $q = (v, \theta)$
end

Figure 88. Overview of a fast pose determination algorithm.

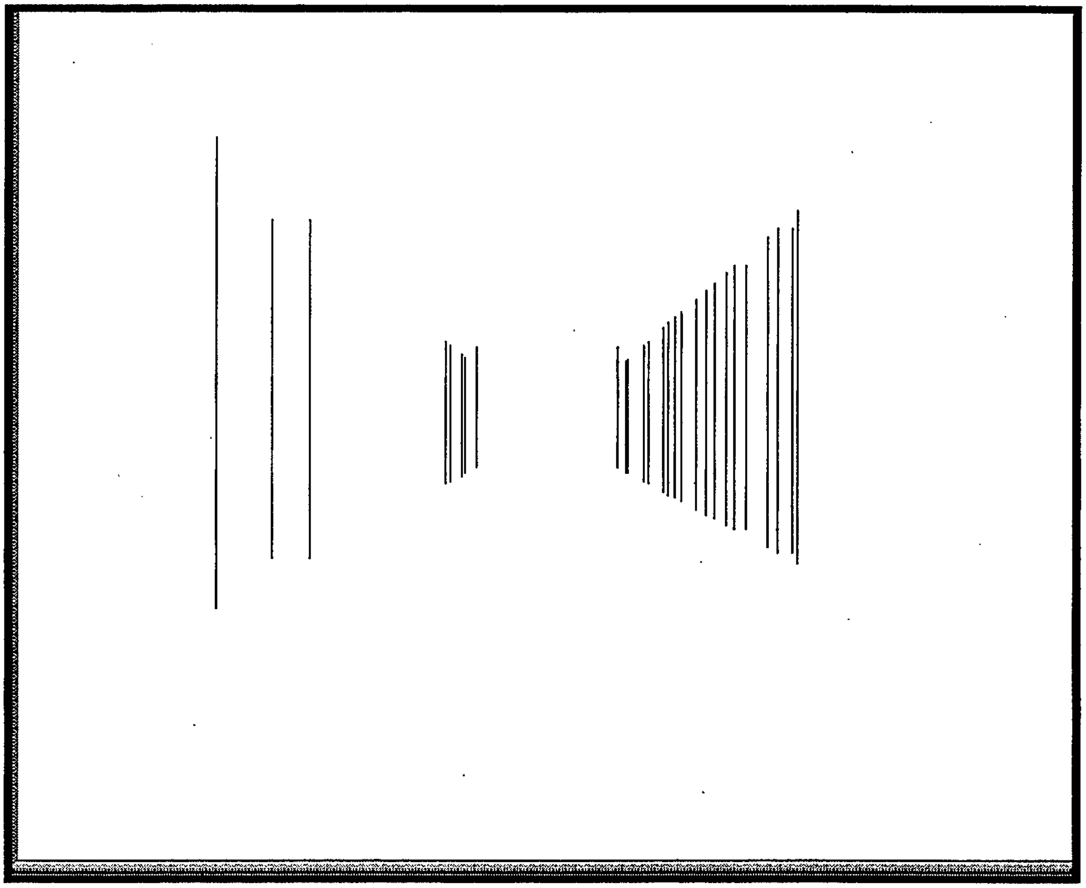


Figure 89. Vertical edges of model expectation 2D view.

To find the correspondence between the features in the $2\frac{1}{2}$D model of the environment and those that exist in the image, it is expected that there exist three vertical edges e_1, e_2, and e_3 in the real world defined by the given model M that best match those three vertical image lines in M_2 (Line 5). The matching process is dressed within the old method in [Ref. 35] and is used in this application.

After obtaining the 2D world coordinates of the positions p_1, p_2, and p_3 on the floor for each of these three model vertical edges (Line 6), the viewing angles from the estimated position to those three points are computed. Let x_1, x_2 and x_3 be the x-coordinates of the intersection point of each vertical image line l_i (for $1 \leq i \leq 3$) and the horizontal projection of the image plane. Furthermore, let x_0 be the x-coordinate of the image plane's center. Since the distance to the image plane from v is the focal

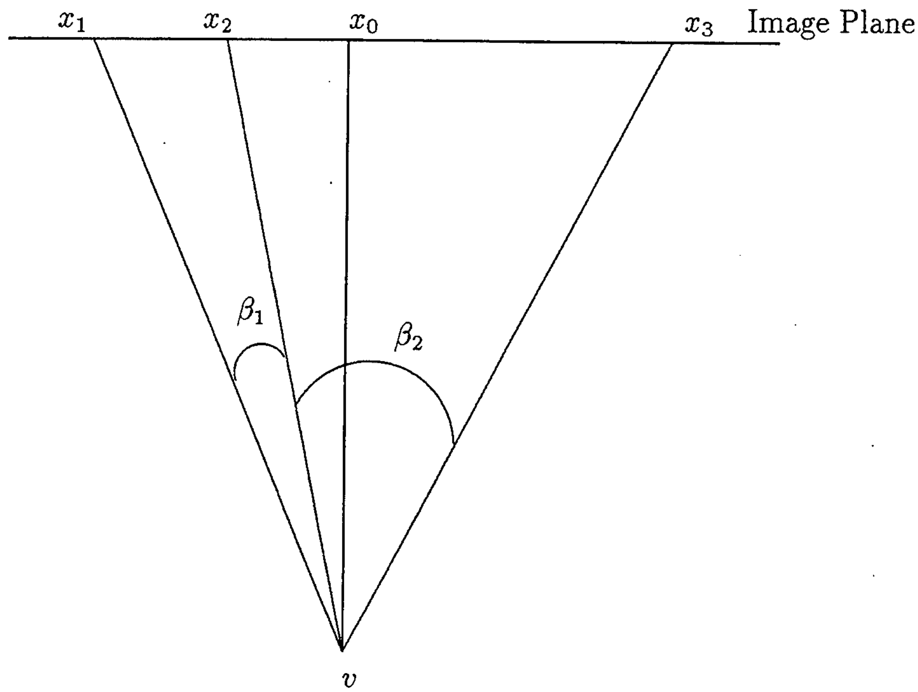


Figure 90. Viewing angles from the camera position v to three positions on image plane lying on the three vertical image lines.

length, the viewing angle β_1 (between image lines l_1 and l_2 from v) and the viewing angle β_2 (between l_2 and l_3 from v) can be computed (Lines 7 and 8) as follows :

$$\begin{aligned}
\beta_1 &= \tan^{-1}\left(\frac{x_2 - x_0}{f}\right) - \tan^{-1}\left(\frac{x_1 - x_0}{f}\right), \\
\beta_2 &= \tan^{-1}\left(\frac{x_3 - x_0}{f}\right) - \tan^{-1}\left(\frac{x_2 - x_0}{f}\right).
\end{aligned}$$

As stated earlier, if p_1, p_2, and p_3 are three control features, then it can be expected that $\beta_1 = \angle p_1 v p_2$ be the viewing angle from the actual camera's position v to p_1 and p_2, and $\beta_2 = \angle p_2 v p_3$ be the viewing angle from v to p_2 and p_3. Given the coordinates of the three control features and the viewing angles β_1 and β_2, the position v can be computed, as we will describe in the next section.

1. Finding Correct Position

The inputs to this task are the three positions p_1, p_2, and p_3 of the world's vertical edges, the computed viewing angles β_1 and β_2, and the estimated pose q_e

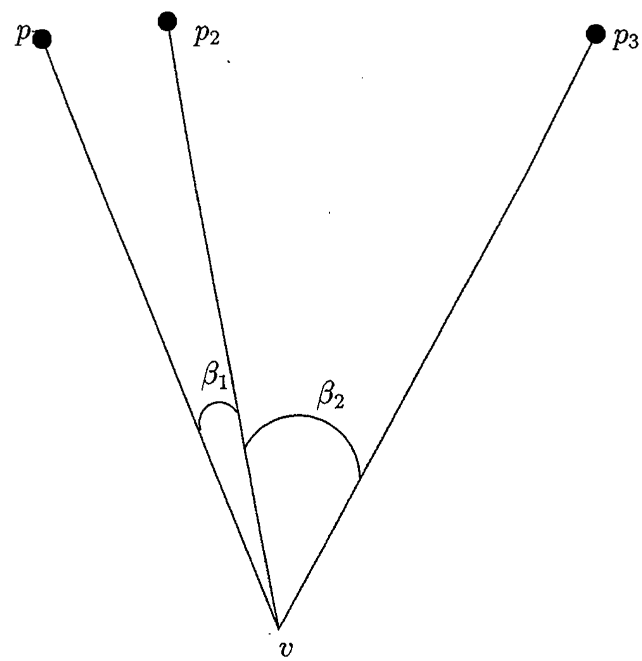


Figure 91. Viewing angles from v to three control features in the world.

(Figure 92). The output is the (x, y) coordinates of the correct position v. To find

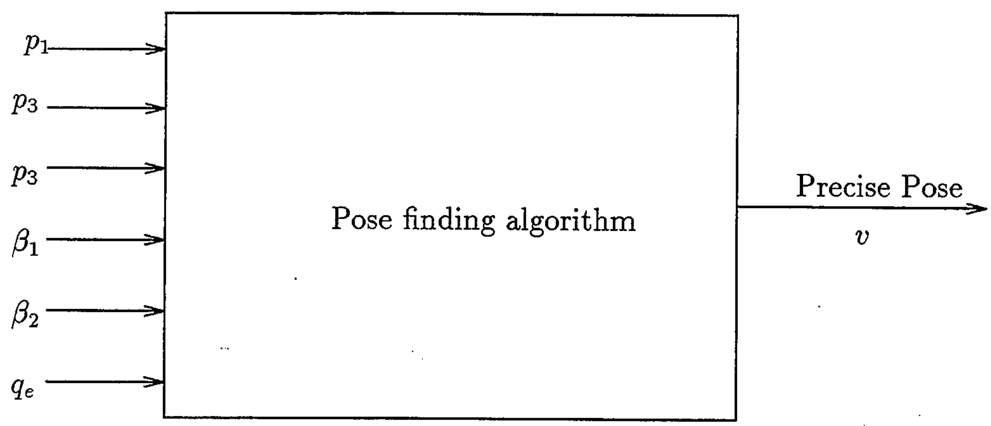


Figure 92. Finding precise position of the camera.

the correct position of v, we will use Figure 93 and the following steps:

1. Let $E_1 = \overline{p_1 p_2}$ with orientation $\alpha_1 = \Psi(p_1, p_2)$, and $E_2 = \overline{p_2 p_3}$ with orientation $\alpha_2 = \Psi(p_2, p_3)$.
2. Compute the center c_1 of the first circle passing through p_1 and p_2 as follows:

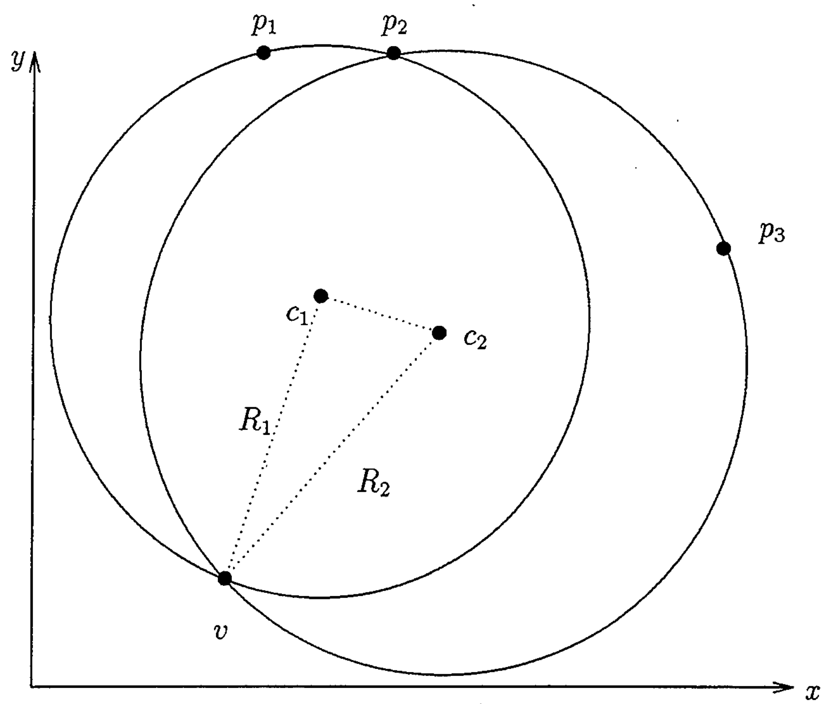


Figure 93. Geometrical relation between the correct position v and the two circles passing through the three control features.

- find the middle point $q_1 = (x_{q_1}, y_{q_1}) = (\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ of $\overline{p_1p_2}$, where $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$.
- compute $s_1 = \frac{\overline{p_1p_2}}{2\tan(\beta_1)}$.
- the center is computed as follows:

$$\begin{aligned} c_1 &= (x_{q_1} + s_1\cos(\alpha_1 - \frac{\pi}{2}), y_{q_1} + s_1\sin(\alpha_1 - \frac{\pi}{2})) \\ &= (x_{q_1} + s_1\sin(\alpha_1), y_{q_1} - s_1\cos(\alpha_1)). \end{aligned}$$

3. Compute the radius R_1 of the first circle:

$$R_1 = \frac{\overline{p_1p_2}}{2\sin\beta_1}.$$

4. Similarly, the second circle's center c_2 and radius R_2 is computed.

5. Given the two centers $c_1 = (x_1, y_1)$ and $c_2 = (x_2, y_2)$, and the two radii R_1 and R_2, the exact position $v = (x, y)$ should satisfy the conditions

$$(x - x_1)^2 + (y - y_1)^2 = R_1^2, \tag{VII.1}$$

and

$$(x - x_2)^2 + (y - y_2)^2 = R_2^2. \tag{VII.2}$$

From these two equations, we get

$$x^2 - 2xx_1 + x_1^2 + y^2 - 2yy_1 + y_1^2 = R_1^2, \tag{VII.3}$$

and

$$x^2 - 2xx_2 + x_2^2 + y^2 - 2yy_2 + y_2^2 = R_2^2. \tag{VII.4}$$

By subtracting Equation VII.4 from Equation VII.3 we get

$$-2x(x_1 - x_2) + x_1^2 - x_2^2 - 2yy_1 + 2yy_2 + y_1^2 - y_2^2 = R_1^2 - R_2^2, \tag{VII.5}$$

which yields

$$\begin{aligned} x &= \frac{x_1 + x_2}{2} - \frac{y_1 - y_2}{2(x_1 - x_2)}y + \frac{y_1^2 - y_2^2}{2(x_1 - x_2)} - \frac{R_1^2 - R_2^2}{2(x_1 - x_2)} && \text{(VII.6)} \\ &= K_1 + K_2 y, && \text{(VII.7)} \end{aligned}$$

where

$$K_1 = \frac{x_1 + x_2}{2} + \frac{y_1^2 - y_2^2 - R_1^2 + R_2^2}{2(x_1 - x_2)}$$

and

$$K_2 = -\frac{y_1 - y_2}{2(x_1 - x_2)}.$$

Solving for Equation VII.3 and substituting for x in the equation, we obtain

$$(K_1 + K_2 y)^2 - 2(K_1 + K_2 y)x_1 + x_1^2 + y^2 - 2yy_1 + y_1^2 = R_1^2,$$

or, equivalently,

$$K_1^2 + 2K_1K_2y + K_2^2y^2 - 2K_1x_1 - 2K_2x_1y + x_1^2 + y^2 - 2yy_1 + y_1^2 - R_1^2 = 0.$$

NoteThis equation is in the form

$$Ay^2 + By + C = 0,$$

where $A = K_2^2 + 1$, $B = 2(K_1K_2 - K_2x_1 - y_1)$, and $C = K_1^2 - 2K_1x_1 + x_1^2 + y_1^2 - R_1^2$.

By computing the constants K_1 and K_2 and using them with the other known values of x_1, y_1, and R_1, we can substitute for A, B, and C in the two solutions of the second order equation which are $y = -B \pm \frac{\sqrt{B^2-4AC}}{2A}$. One of the values of y should be very close to that of the point p_2, so the other value is taken. Using that value of y in Equation VII.7, x can be computed. Now the correct position is $v = (x, y)$.

2. Finding Correct Orientation

After finding the correct position $v = (x, y)$, the correct orientation, θ, is computed as follows:

For each control feature p_i, $(1 \leq i \leq 3)$, the orientation $\Psi(v, p_i)$ is computed. Next, the angle δ_i is computed using the focal length, x_i-values for each corresponding vertical line on the image plane, and x_0, the x-coordinate of the image plane's center (Figure 94). Then the difference $\theta_i = \Psi(v, p_i) - \delta_i$ is computed. Finally, θ is obtained by averaging these values. The algorithm is given in Figure 95.

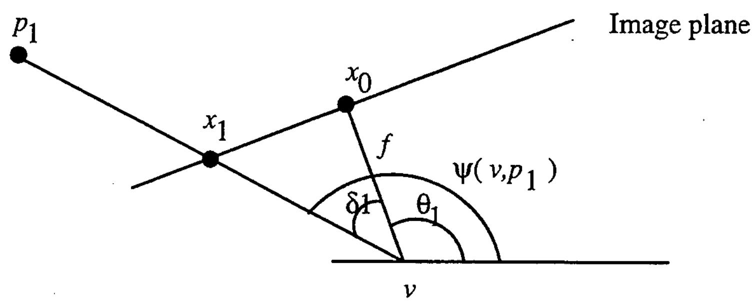


Figure 94. Geometry of finding camera orientation.

E. EXPERIMENTAL RESULTS

Using the model of the new environment of the autonomous robot Yamabico, the algorithm was tested on some images taken by a CCD camera. We estimated the position and orientation in which each image was taken. Several experiments were performed to verify that the algorithm returns the correct pose. The difficult part was specifying the focal length to obtain the correct model view that will be matched

CorrectOrientation$(f, v, p_1, p_2, p_3, x_0, x_1, x_2, x_3)$

begin
for (each $p_i, 1 \leq i \leq 3$)
    $\Psi_i = \text{Orientation}(v, p_i)$
    $\delta_i = \tan^{-1}\left(\frac{x_i - x_0}{f}\right)$
    $\theta_i = \Psi_i - \delta_i$
return $\theta = \text{Average}(\theta_1, \theta_2, \theta_3)$
end

Figure 95. Algorithm of computing the correct camera orientation θ.

with the actual image. In Figure 97, we show an example of correcting the estimated pose in Figure 96.

Figure 96. Estimated pose x=2040 cm, y= 120.0cm, $\theta = 0.0°$.

The magnitude of the error for that position was 10.77 cm for x, 3.77 cm for y and 1.97° for θ. The execution time on a 20MHz SGI machine was about 4-5 seconds which is about 20% of the execution time of the old pose determination method. Thus, we believe that a significant improvement was achieved by using the

Figure 97. Correct pose x=2050.77 cm, y= 116.23, $\theta = 1.97°$.

new efficient edge detection algorithm and the modeling method.

VIII. IMPLEMENTATION PLATFORM: YAMABICO-11

In this chapter, a brief description of the hardware and software system architecture of the robot Yamabico-11 is given. The on-board image understanding system is also described here. This is intended as background information about the platform on which the research in this dissertation was directed.

A. YAMABICO HARDWARE SYSTEM

The autonomous mobile robot Yamabico-11 (Figure 98) is an experimental robot used for indoor robotics research at the Naval Postgraduate School. The efforts

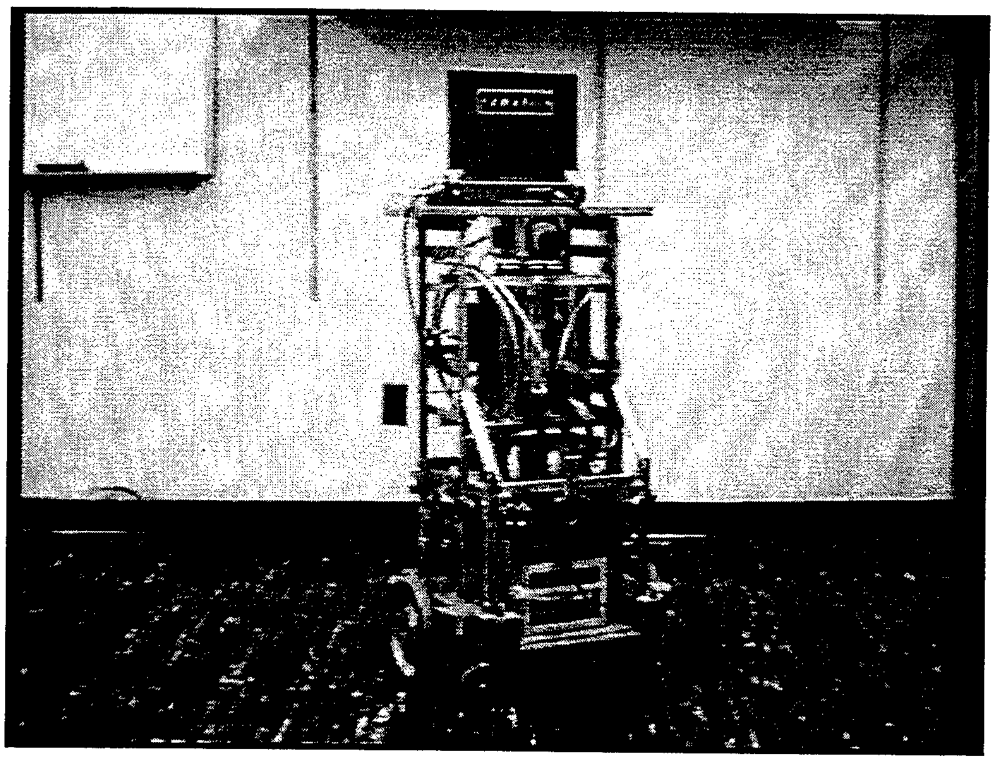


Figure 98. Autonomous mobile robot, Yamabico-11.

of a research team led by Professor Yutaka Kanayama have added many significant contributions to that platform over several years. Many research activities have been involved in the project, including motion planning [Ref. 9, 10, 11], automated cartography [Ref. 33], sonar-data interpretation [Ref. 18, 19], and sonar-based obstacle avoidance [Ref. 34]. Additional contributions have been made at the University of Electro-Communications, the University of Tsukuba, Stanford University, and the University of California at Santa Barbara [Ref. 76, 77].

For image understanding, several efforts constructed functions that can be used for further image understanding tasks ([Ref. 35, 36]). A vision system was added to the robot, as described in [Ref. 38]. The Yamabico hardware system is described in Table V. Some of the technical information in this table is based on [Ref. 78]. The hardware architecture of Yamabico is illustrated in Figure 99.

Power Source	Two 12-volt batteries
CPU	IV-SPARC, 33MHz
RAM	16 MB DRAM
MIPS	24
MFLOPS	5
CPU Interface	VMEbus Interface
EPROM	Two 64K $\times$ 16 EPROMs
Driving Wheels	2
Free Wheels	Four casters (for balance)
Communication With UNIX System	Ethernet (10Base-T)
Sonar System	Twelve 40KHz sonars (3 groups)
Sonar Beam Range	4 meters (approximately)
Sonar Coverage	360 degree
Motors	Two 35 watt DC motor with shaft encoders
Motor Control	Dual Axis Controller
Vision System	image board with CCD camera (B&W)
I/O Interface With Users	Macintosh PowerBook

Table V. Yamabico' s main hardware specifications.

B. ON-BOARD VISION SYSTEM

In this section we give a brief description of the on-board vision system. The installation of an image board and a CCD camera on the robot was an important step in Yamabico's development. That was the initial step in the on-board image

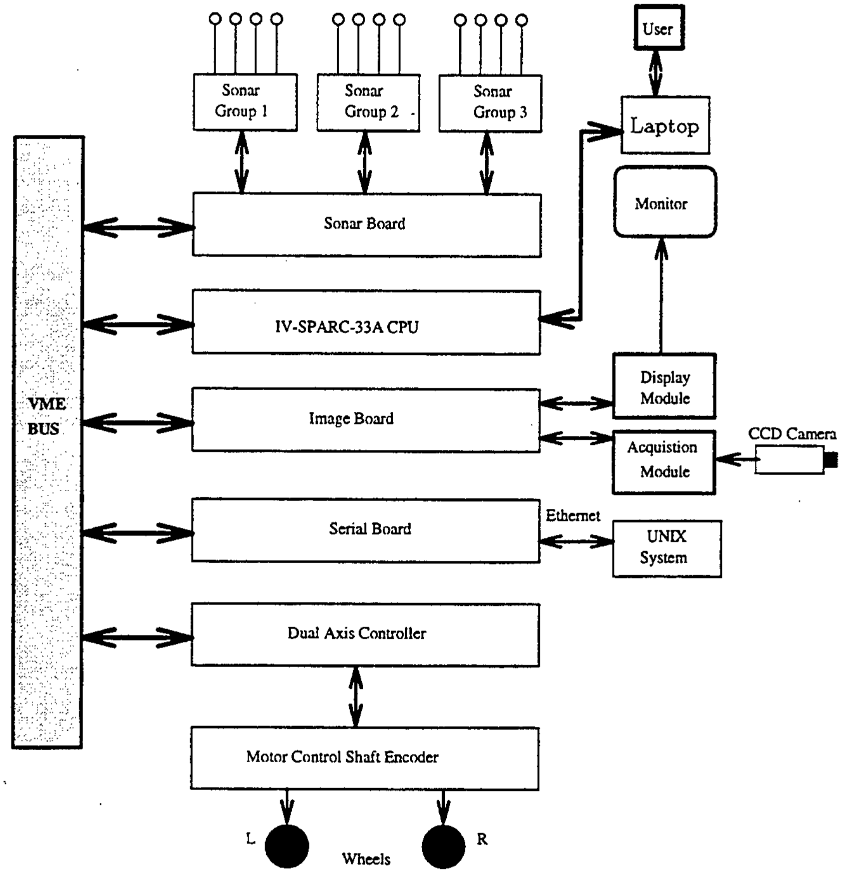


Figure 99. Block diagram of Yamabico-11 hardware architecture.

understanding system. The efforts of J. Kisor [Ref. 38] in integration of image hardware/software became the groundwork for the future image understanding tasks. The main references for this section were his thesis and the technical manuals of the image board and the camera used.

1. IMS Image Board

The IMS (standard image manager) is a single VME board containing several components such as image memory, acquisition module (AM), computational module (CM) and display module (DM). These modules provide flexible camera and display interfaces. Additionally, they provide local processing capabilities. The acquisition module provides an interface between the camera and the IMS board. The display module provides an interface with a display monitor [Ref. 79]. The image board is mapped to the addressing space of the VME bus as shown in Table VI. The IMS

Module	Address
Image Memory Frames or Display Module	0xfa000000
IMS registers	0xfc000600

Table VI. Mapping of image modules to the VME bus address space.

image memory stores digitized images at three frames; A0, A1, and B1. Each frame is 1024 pixels × 1024 lines, with an eight-bit per pixel.

2. CCD Camera

The image understanding system on Yamabico includes a Cohu 4110 CCD camera from Cohu Inc [Ref. 80]. The Cohu 4110 is a digital output camera featuring a 1/2 inch format CCD image sensor. The area of active imaging is 6.4mm × 4.8mm and 739 × 484 pixels. Table VII shows the main characteristics of the Cohu camera. For more information see [Ref. 80].

imager	single CCD
Image Sensor Size	6.4mm × 4.8mm
Video Output	digital with analog option
Resolution	739 × 484 pixels
Num. of bits/pixel	8 bits
Gray Levels	256
Focal Length	variable (depends on the lens used)

Table VII. Summary of Cohu CCD camera capabilities.

C. MML SOFTWARE

The Model-based Mobile-robot Language (MML) is a library of functions written in the ANSI C language in the UNIX environment. This library is used as a basis for all user programs and commands to the robot. It supports motion control functions, sonar functions, I/O functions, image understanding functions, and others. The last version developed for Yamabico is called MML-11. A new version (MML-12) for motion control is currently under development. The image understanding portion of the software was integrated with MML-11 for experimental purposes [Ref. 38]. In the future, it should be integrated with the new version MML-12.

When a user writes a program and wants to execute it on Yamabico, the program takes the form of a user file called *user.c* which includes some commands to the robot that should be performed sequentially. After compiling the program, the executable file is ready to be downloaded to the robot through an Ethernet connection. After downloading the program, the robot becomes self-contained.

The function library is a set of *user functions* to provide the interface between the user and the system. Before adding the image understanding functions, the user functions were categorized into four modules [Ref. 11]:

- Operating System Module,
- Motion Planning Module,
- Motion Control Module,
- Sonar Control Module.

An important feature in the software system of Yamabico is its interrupt handling. The motion control has the highest priority. The interrupt occurs every 10 msec. The sonar control can issue an interrupt request every 50 msec in the case of the existence of sonar returns. The user programs have the lowest priority, but it gives the system all the requested user commands through a sequential buffer.

D. IMAGE UNDERSTANDING SOFTWARE

The image understanding software was added first to the robot as a part of the work addressed in [Ref. 38]. Some low-level image functions were included with the system through that work. In the following, we list those main low-level image functions:

1. IMS initialization : for initializing the image board.
2. Display Initialization : for setting the monitor and display module registers.

3. Acquisition Initialization: for setting registers that control the camera or acquisition module interface.

4. Image Transfer (setInputPath, setFrameAcquire, acqEnable, and interlacePage2).

1. Basic User-Level Image Functions

Adding an image understanding capability to the robot requires the addition of an image function library. It is intended to hide the low-level functions from the user, who may then issue user-level commands in place of a sequence of low-level functions. These low-level functions were originally designed by John Kisor [Ref. 38]. Specifically, the functions acqEnable and interlacePage2 can be categorized in the user-level ones. All the user-level commands are contained in the Appendix: ON-BOARD IMPLEMENTATION, Part 4. The added user-level commands are shown in Table VIII.

Function	Purpose
initImgSys()	Initialization of the image frames
setCamera()	set the input path from the camera to the 3 image frames
startGrab()	direct the image board to start grabbing continuous video image from the camera to image frames
stopGrab()	direct the image board to stop grabbing the video image
snap()	take a snapshot of a still image into image frames
clearFrame(frame)	set all pixel data in a specified frame to 0
clearFrames()	set all pixel data in *all* frames to 0

Table VIII. Summary of basic user image functions.

IX. CONCLUSIONS AND DIRECTIONS FOR FUTURE WORK

A. CONCLUSIONS

This research has solved the efficiency problem for the model-based image understanding tasks of autonomous vehicles. A new efficient straight-edge detection method using *direction-controlled edge tracking* and *random hitting* algorithms was invented for the purpose of reducing the computational time. This method avoids the exhaustive pixel processing executed in classical edge detection methods. We proposed using a pseudo-random number scheme to find a starting pixel from which an edge tracking for a linear sequence of pixels is performed. A robust least-squares fitting algorithm was used to obtain the geometric features for the line segment that best fits the pixel sequence including the endpoints. The directional information of the line segment was used to control the edge tracking task. During this edge detection method, duplicate tracking of the same edge was avoided by the closeness test.

In real experiments, we were able to detect 20 major edges in a test image with a very small number of pixels processed (4.36 % of 313,956 total pixels in the image). The results show that this algorithm is useful and efficient for numerous image understanding applications and autonomous robot visual navigation.

An effective method for modeling the world environment of an indoor autonomous vehicle is presented to give the main features of the world in a simple and efficient manner. The two methods were used to improve the efficiency of the pose determination algorithm. The execution time was only about 20-25% of that with a 3D model and exhaustive scanning method for edge detection.

These methods were first successfully implemented on a graphics workstation. Second, the code was partially ported to the autonomous mobile robot Yamabico-11. A user-level image function library was added to the current MML software system, so that a user can easily program high-level on-board vision algorithms.

B. FUTURE WORK

In the following points, some suggestions for future work:

- Investigating different values to be used for c in the pseudo-random hitting method that satisfies the uniform-distribution property, and finding a mathematical model that computes the best scheme for the pseudo-random number generator, based on the image width and height.
- Investigating the use of the new edge detection method with multi-resolution images, as those dealt with in [Ref. 81].
- Implementing the $2\frac{1}{2}$D model and pose determination in real time for self localization.
- Including the presented direction-controlled edge-tracking method with an obstacle avoidance method may result in a robust, intelligent behavior of the robot.
- Investigating sonar/vision fusion to verify the localization correction.
- There were some minor problems in edge detection in some indoor images due to the reflection and lighting conditions. Solving this problem will improve this edge-tracking method.
- Complete implementation of the image-understanding function library.

APPENDIX A. ON-BOARD IMPLEMENTATION

The new edge detection algorithm was implemented on the autonomous mobile robot Yamabico-11. We present the data structures and the program design as implemented on the robot Yamabico. The program design is described in Figure 100.

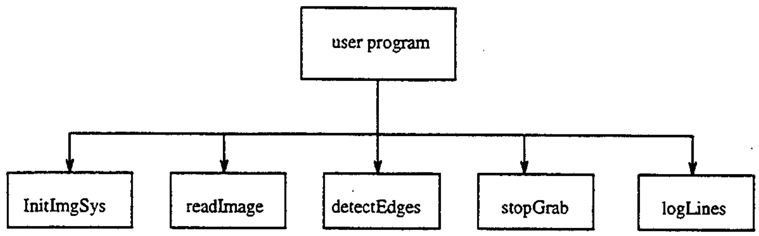


Figure 100. User program description.

In this user program, a call to the function **initImgSys** performs initialization of the image frames A0, A1, B1 to zero. The function **readImage** sets the camera signals to be the input to the image board, then starts grabbing an image into frames A0, A1, and B1. After setting the appropriate memory location for the acquired image, a call to the function detectEdges() is applied on the image to perform the edge detection task. The structure of readImage() function is described in Figure 101, while the structure of detectEdges is shown in Figure 102.

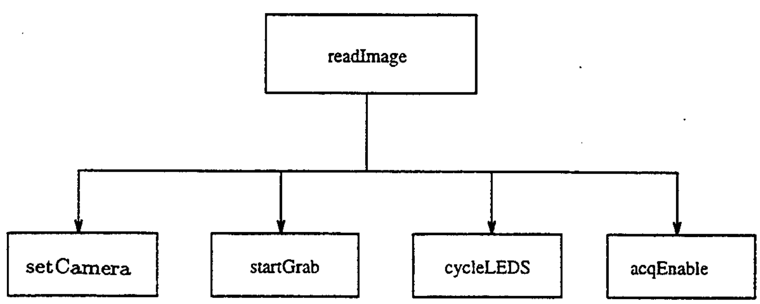


Figure 101. Structure of the readImage() function.

An example of a user program is given below, followed by the data structures and the edge detection functions.

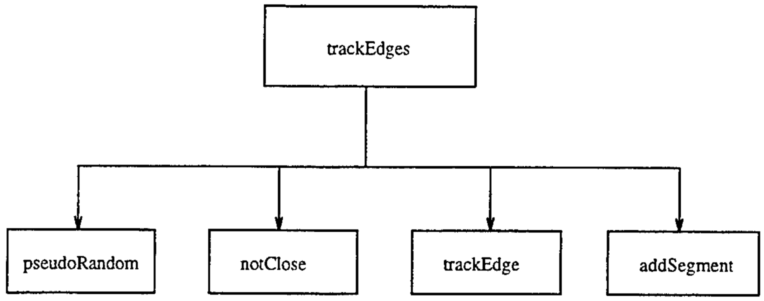


Figure 102. Edge detection program description.

1. AN EXAMPLE OF USER PROGRAM

```
/*****************************************************
FILENAME    : user.c
DESCRIPTION: read image and detect edges using
             Direction Controlled Edge Tracking and
             Random Hitting
Author      : Khaled Morsy
*****************************************************/
#include "user.h"
#include "displayCtrl.h"
#include "acquisitionCtrl.h"
#include "imsControl.h"
#include "imgLib.h"
#include "trackEdge.h"

/**** Local Prototypes *****/

int
user(){

   initImgSys();
   readImage();
   detectEdges();
   stopGrab();
   logLines();
}
```

2. DATA STRUCTURES DEFINITIONS

```
/*****************************************************
FILE NAME  : imgDefs.h
DESCRIPTION: Data structures Definitions for image
             understanding programs, specifically for the
             new edge detection algorithm.
AUTHOR     : Khaled Morsy
*****************************************************/

typedef struct
{
  POINT p;
  double gm;
  double phi;
} PIXEL;

typedef struct
{ PIXEL p1;
  PIXEL p2;
  PIXEL p3;
} GROUP;

typedef struct
{
  /* Least Squares Fit moments*/
  double m00;       /* Number of pixels */
  double m10;       /* Sum x */
  double m01;       /* Sum y */
  double m11;       /* Sum x*y */
  double m20;       /* Sum x*x */
  double m02;       /* Sum y*y */
  double mux ;      /* x-value of centroid */
  double muy ;      /* y-value of centroid */
  double alpha;     /* calculated normal orientation
                       of IMG LINE */
  double r;         /* distance from origin to the line
                       segment */
  POINT  q1;        /* last pixel of edge sequence in psi+
                       direction */
  POINT  q2;        /* last pixel of edge sequence in psi-
                       direction */
} EDGE;
```

```
typedef struct
{ double n;          /* same as m00  */
  double alpha;
  double r;
  POINT e1;
  POINT e2;
} IMG_LINE;
```

3. FUNCTIONS OF THE DIRECTION-CONTROLLED EDGE TRACKING WITH RANDOM HITTING PROGRAM

```
/*****************************************************
FILE NAME    : trackEdge.h
PURPOSE      : Edge detection functions
DATE         : Apr. 9, 1997
Notes        : Yamabico Version.
AUTHOR       : KHALED MORSY
*****************************************************/
#define THRESHOLD 5000
#include "imgDefs.h"

/***                    Global Variables            */

double phi0 , g0, phi1;  /* p0's gradient direction and
                            magnitude */

int temp = 1;
int H = 476;
int W = 732;

/*************************** PROTOTYPE **************/
long Pseudorandom(long,long,long,long);
POINT ConvertTo2D(long);
int NotOnImageBoundary(POINT);
int significant(POINT);
void ComputeGradient(POINT, double *, double * );
double SumSqrs(double , double);
POINT FindInitialPixel(POINT);
void TrackEdge(POINT,EDGE  ,IMG_LINE *);
void InitializeSequence(POINT , EDGE *);
double normalize2(double) ;
void TrackPixels(POINT,double,EDGE *);
POINT SelectNextPixel(POINT,double);
GROUP Neighbors(POINT , double);
PIXEL  LargestGradientPixel(GROUP);
int  Consistent(double ,double );
void DeletePixel(PIXEL, GROUP*);
void LeastSquaresFitting(EDGE *,POINT);
void ComputeSegmentData(EDGE * , IMG_LINE *);
double normalize(double) ;
```

```
void InitLogLines(char*, int);

/***************************************************
Function : PseudoRandom(long r0, long c , long M)
Purpose  : Generate a Pseudo Random Number
AUTHOR   : Khaled Morsy
***************************************************/
long PseudoRandom(long r0, long a, long c, long M)
{
  return (a*r0 + c) % M;
}

/***************************************************
Function : ConvertTo2D(long)
Purpose  : converts a Pseudo Random Number into image
           coordinates (x,y)
AUTHOR   : Khaled Morsy
***************************************************/
POINT ConvertTo2D(long prn)
{
   POINT p;
   p.X= prn % W ;
   p.Y = prn / W ;
   return p;
}

/***************************************************
Function : NotOnImageBoundary(POINT)
Purpose  : true if the point NOT on the image boundary
AUTHOR   : Khaled Morsy
***************************************************/
int NotOnImageBoundary(POINT p)
{
  if((p.X > 2 && p.X < W-2) && (p.Y > 2 && p.Y < H-2))
    return 1 ;
  else
    return 0 ;
}

/***************************************************
Function : Significant(POINT p0)
Purpose  : returns true if p0 is a significant pixel
```

```
           and computes its gradient direction
AUTHOR    : Khaled Morsy    Jan 97
*****************************************************/
int significant(POINT p0 )
{
  double gx,gy ;
  double g;
  if(NotOnImageBoundary(p0))
  {
     ComputeGradient(p0,&gx,&gy);
     g0 =  SumSqrs(gx,gy);
     if (g0 > THRESHOLD)
     {  phi0 = atan2(gy,gx);
        return 1 ;
     }
     else return 0;
  }
   return 0 ;
}
/*****************************************************
Function : ComputeGradient(POINT p0,gx,gy)
Purpose  : compute gx  and gy (horizontal and vertical
           gradients) by SOBEL OPERATOR
AUTHOR   : Khaled Morsy    Jan 97
/*****************************************************/
void ComputeGradient(POINT p, double *gx , double *gy)
{

  /* this deals with frame B1 */
  int x,y,gxi,gyi;
  BYTE ul,l,dl,u,d,ur,r,dr;
  unsigned long p0,b = 0xfa000000;;

  x=(int) p.X;
  y=(int) p.Y;
  p0 = b + y * 1024 + x ;

  /* the direction of pixels on-board are different from
     SGI machine, it is left to right top to bottom
     a pixel and the one on the next line differs by
     1K */
```

```
  l = *(BYTE*) (p0-1);      /* left pixel  */
  r = *(BYTE*) (p0+1);      /* right pixel */
  u = *(BYTE*) (p0-1024);  /* upper pixel */
  ul = *(BYTE*)(p0-1025);  /* upper-left  */
  ur = *(BYTE*)(p0-1023);  /* upper-right */
  d = *(BYTE*) (p0+1024);  /* down pixel  */
  dl = *(BYTE*)(p0+1023);  /* down-left   */
  dr = *(BYTE*) (p0+1025); /* down-right  */

  gxi = -ul-l-dl+ur+r+dr;
  gyi = -dl-d-dr+ul+u+r;

  *gx = (double) gxi ;

  *gy = (double) gyi ;
}


/*****************************************************
Function : SumSqrs(a,b)
Purpose  : simply compute a*a + b*b
AUTHOR   : Khaled Morsy
*****************************************************/
double SumSqrs(double a, double b)
{
  return a*a + b*b ;
}

/****************************************************
FUNCTION : TrackEdge(PIXEL p0 , EDGE *Q ,IMG_LINE *L)
PURPOSE  : track pixel sequence in two directions
AUTHOR   : Khaled Morsy
*****************************************************/
void TrackEdge(POINT p0 , EDGE Q ,IMG_LINE *L)

{
  InitializeSequence(p0, &Q);

  TrackPixels(p0, PI/2, &Q);

  TrackPixels(p0, -PI/2, &Q);
```

```
  ComputeSegmentData(&Q,L);

 }

/*******************************************************
FUNCTION :  InitializeSequence(POINT p, EDGE *Q)
PURPOSE  :  Initailize Least-Squares fitting parameters
AUTHOR   :  Khaled Morsy
*******************************************************/
void InitializeSequence(POINT p, EDGE *Q)
{
  Q->m00   = 1;
  Q->m10   = p.X;
  Q->m01   = p.Y;
  Q->m20   = p.X * p.X ;
  Q->m02   = p.Y * p.Y ;
  Q->m11   = p.X * p.Y ;
  Q->alpha = normalize2(phi0) ;
  Q->r     = 0.0 ;
  Q->q1.X  = p.X ;
  Q->q1.Y  = p.Y ;
  Q->q2.X  = p.X ;
  Q->q2.Y  = p.Y ;
}

/*******************************************************
FUNCTION : normalize2()
PURPOSE  : return the normalized value of orientation
           in range -PI/2 , PI/2
AUTHOR   :  Khaled Morsy
********************************************************/

double normalize2(double o)
{
while ( o > PI/2)  o = o - PI ;
while ( o < - PI/2 ) o = o + PI ;
return(o);
}

/*******************************************************
FUNCTION : TrackPixels(POINT p0, double beta, EDGE *Q)
PURPOSE  : track only one side from p0
```

```
AUTHOR    : Khaled Morsy
*****************************************************/

void TrackPixels(POINT p0, double beta, EDGE *Q)
{
  POINT p1,p2;
  double psi;
  p1=p0;
  phi1=phi0;
  while(1)
  {
      psi = Q->alpha + beta ;
      p2 = SelectNextPixel(p1,psi);
      if(p2.X == 0.0 && p2.Y == 0.0)
          {
            if(beta == PI/2)
              { Q->q1.X = p1.X ;
                Q->q1.Y = p1.Y ;}
            else
              { Q->q2.X = p1.X ;
                Q->q2.Y = p1.Y ;
              }

            break;
          }

      LeastSquaresFitting(Q,p2);
      p1=p2 ;

  }
}

/*****************************************************
FUNCTION : SelectNextPixel(POINT p1, double psi)
PURPOSE  : select one pixel out of 3 neighbors
AUTHOR   : Khaled Morsy
/*****************************************************/

POINT SelectNextPixel(POINT p1,double psi)
{
  PIXEL pxl2;
  double phi2;
```

```
  int num_pixels_in_group = 3;
  GROUP S;
  POINT p2;
   S = Neighbors(p1, psi);
    while(num_pixels_in_group > 0)
    {
      pxl2 = LargestGradientPixel(S);

      if(pxl2.p.X != 0.0 && pxl2.p.Y != 0.0 )
      {  phi2 = pxl2.phi;
         if(NotOnImageBoundary(pxl2.p) && Consistent(psi,phi2))
    return pxl2.p;
      }
      DeletePixel(pxl2,&S); /* make gm=0 of corresp. pixel to
                                 pxl2 in S */
      num_pixels_in_group-- ;
    }
   p2.X = 0.0;
   p2.Y = 0.0;
   return(p2);
}

/****************************************************
FUNCTION : Neighbors(POINT p, double psi)
PURPOSE  : find neighbors of a given point in the psi
           direction.
AUTHOR   : Khaled Morsy
/****************************************************/

GROUP Neighbors(POINT p, double psi)
{
  GROUP S;
  POINT pp1,pp2,pp3;
  double gx,gy,gm;

   if(psi < - PI)
     psi = -PI ;

  if(psi > -PI/8 && psi <= PI/8)     /* range D0 */
     {
      pp1.X = p.X + 1 ; pp1.Y = p.Y -1;
      pp2.X = p.X + 1 ; pp2.Y = p.Y ;
```

```
        pp3.X = p.X + 1 ; pp3.Y = p.Y +1 ;


      }
    else if(psi > PI/8 && psi <= 3*PI/8)     /* range D1 */
       {
         pp1.X = p.X + 1 ; pp1.Y = p.Y ;
pp2.X = p.X + 1 ; pp2.Y = p.Y + 1 ;
         pp3.X = p.X   ;  pp3.Y = p.Y + 1 ;
       }
    else if(psi > 3*PI/8 && psi <= 5*PI/8)     /* range D2 */
       {
           pp1.X = p.X + 1 ; pp1.Y = p.Y + 1 ;
  pp2.X = p.X   ;  pp2.Y = p.Y + 1 ;
           pp3.X = p.X -1 ;  pp3.Y = p.Y + 1 ;
        }


    else if(psi > 5*PI/8 && psi <= 7*PI/8)     /* range D3 */
       {
           pp1.X = p.X   ; pp1.Y = p.Y + 1 ;
  pp2.X = p.X-1 ;  pp2.Y = p.Y + 1 ;
           pp3.X = p.X -1 ;  pp3.Y = p.Y   ;
        }

  else if((psi > 7*PI/8 && psi <= PI) ||
           (psi >= -PI && psi <= -7*PI/8))     /* D4 */
       {
           pp1.X = p.X-1  ; pp1.Y = p.Y + 1 ;
  pp2.X = p.X-1 ;  pp2.Y = p.Y   ;
           pp3.X = p.X -1 ;  pp3.Y = p.Y -1   ;
        }
      else if(psi > -7*PI/8 && psi <= -5*PI/8)  /* range D5 */
       {
         pp1.X = p.X - 1 ; pp1.Y = p.Y   ;
         pp2.X = p.X - 1 ; pp2.Y = p.Y -1 ;
         pp3.X = p.X   ;  pp3.Y = p.Y - 1 ;
       }

           else if(psi > -5*PI/8 && psi <= -3*PI/8) /* range D6 */
       {
         pp1.X = p.X - 1 ; pp1.Y = p.Y-1   ;
```

```
    pp2.X = p.X  ; pp2.Y = p.Y -1 ;
    pp3.X = p.X + 1 ;  pp3.Y = p.Y - 1 ;
  }

  else if(psi > -3*PI/8 && psi <= -PI/8)  /* range D7 */
  {
    pp1.X = p.X     ; pp1.Y = p.Y-1  ;
    pp2.X = p.X + 1 ; pp2.Y = p.Y -1 ;
    pp3.X = p.X + 1 ; pp3.Y = p.Y ;
  }

  S.p1.p.X = pp1.X; S.p1.p.Y = pp1.Y ;
  S.p2.p.X = pp2.X; S.p2.p.Y = pp2.Y ;
  S.p3.p.X = pp3.X; S.p3.p.Y = pp3.Y ;
  ComputeGradient(pp1,&gx,&gy);
  gm = SumSqrs(gx,gy);
  if(gm > THRESHOLD)
    {S.p1.gm = gm ;
     S.p1.phi = atan2(gy,gx);
    }
  else
{S.p1.gm = 0.0;
     S.p1.phi = 0.0;
    }

  ComputeGradient(pp2,&gx,&gy);
  gm = SumSqrs(gx,gy);
  if(gm > THRESHOLD)
    {S.p2.gm = gm ;
     S.p2.phi = atan2(gy,gx);
    }
  else
{S.p2.gm = 0.0;
     S.p2.phi = 0.0;
    }


  ComputeGradient(pp3,&gx,&gy);
  gm = SumSqrs(gx,gy);
  if(gm > THRESHOLD)
    {S.p3.gm = gm ;
     S.p3.phi = atan2(gy,gx);
```

```
        }
      else
{S.p3.gm = 0.0;
         S.p3.phi = 0.0;
        }
  return S;
}

/***************************************************
FUNCTION : LargestGradient(GROUP *S)
PURPOSE  : select one pixel with largest gradient
           from group S of pixels
AUTHOR   : Khaled Morsy
/***************************************************/

PIXEL LargestGradientPixel(GROUP S)
{
   PIXEL pxl;
   double gm1,gm2,gm3;
   gm1 = S.p1.gm ;
   gm2 = S.p2.gm ;
   gm3 = S.p3.gm ;

   if( gm1 > THRESHOLD &&(gm1 > gm2 && gm1 > gm3) )
     {
      if(gm1 - gm2 > (.1 * gm2))
         return S.p1;
      else
        return  S.p2 ;

     }

   else if(gm2 > THRESHOLD &&(gm2 >= gm1 && gm2 >= gm3))
        return S.p2;

   else if(gm3 > THRESHOLD && (gm3 > gm1 && gm3 > gm2))
    {
      if(gm3 - gm2 > (.1 * gm2))
        return  S.p3;
      else
        return S.p2 ;
    }
```

```
    else
      {
          pxl.p.X = 0.0 ;
          pxl.p.Y = 0.0 ;
          pxl.gm = 0.0 ;
          pxl.phi = 0.0 ;
     return pxl ;

      }

}

/**************************************************
FUNCTION : Consistent(double phi ,double phi)
PURPOSE  : check consistency between two directions
AUTHOR   :  Khaled Morsy
/**************************************************/
int Consistent(double psi ,double phi)
{
  double epsilon = 20 * PI / 180;
  if(fabs(normalize2(phi-psi-PI/2))  < epsilon )
      return(1) ;
  else
      return(0);
}

/***************************************************
FUNCTION : Delete(PIXEL pxl, GROUP S)
PURPOSE  : make grad. magnitude of pxl zero, so it will
           not be used next time
AUTHOR   :  Khaled Morsy
/***************************************************/

void DeletePixel(PIXEL pxl , GROUP *S)
{

    if( pxl.p.X == S->p1.p.X && pxl.p.Y == S->p1.p.Y)
       S->p1.gm = 0.0 ;
    else if(pxl.p.X == S->p2.p.X && pxl.p.Y == S->p2.p.Y)
        S->p2.gm = 0.0 ;
    else
        S->p3.gm = 0.0 ;
```

```
}

/**************************************************
FUNCTION :  LeastSquaresFitting(EDGE Q,  POINT p)
PURPOSE  :  Compute The Least-Squares fitting for pixel
            sequence Q
AUTHOR   :  Khaled Morsy
**************************************************/
void LeastSquaresFitting(EDGE *Q,  POINT p)
{
  double M20, M11, M02;

  ++Q->m00 ;
  Q->m10 += p.X;
  Q->m01 += p.Y;
  Q->m20 += p.X * p.X ;
  Q->m02 += p.Y * p.Y ;
  Q->m11 += p.X * p.Y ;
  M20 = Q->m20 - (Q->m10)*(Q->m10)/Q->m00 ;
  M11 = Q->m11 - (Q->m10)*(Q->m01)/Q->m00 ;
  M02 = Q->m02 - (Q->m01)*(Q->m01)/Q->m00 ;
  Q->alpha = 0.5 * atan2(-2*M11 , M02-M20);
}

/**************************************************
FUNCTION : ComputeSegmentData(EDGE *Q, IMG_LINE *L)
PURPOSE  : construct Line Segment data struct.
AUTHOR   :  Khaled Morsy
/*************************************************/
void ComputeSegmentData(EDGE *Q ,IMG_LINE *L )
  {
      double delta;
      L->n = Q->m00 ;
      L->alpha = Q->alpha ;
      L->r = Q->m10/Q->m00 * cos(Q->alpha) +
             Q->m01/Q->m00 * sin(Q->alpha) ;
      delta = Q->q1.X * cos(Q->alpha) +
              Q->q1.Y * sin(Q->alpha) - L->r ;
      L->e1.X = Q->q1.X - delta * cos(Q->alpha);
      L->e1.Y = Q->q1.Y - delta * sin(Q->alpha);
      delta = Q->q2.X * cos(Q->alpha) +
```

```
            Q->q2.Y * sin(Q->alpha) - L->r ;
      L->e2.X = Q->q2.X - delta * cos(Q->alpha);
      L->e2.Y = Q->q2.Y - delta * sin(Q->alpha);
}
/******************************************************
FUNCTION : AddSegment(LS, L, i)
PURPOSE  : add one segment to the array
AUTHOR   : Khaled Morsy
******************************************************/
void AddSegment(LS, L, i)
 IMG_LINE LS[]; IMG_LINE L;int i;

{
        LS[i].n = L.n;
        LS[i].alpha = L.alpha;
              LS[i].r = L.r ;
              LS[i].e1 = L.e1 ;
              LS[i].e2 = L.e2 ;
}

/*****************************************************
FUNCTION : NotClose(p,L,index)
PURPOSE  : Closeness Test
AUTHOR   : Khaled Morsy
*****************************************************/
int NotClose(p,L,index)
POINT p ; IMG_LINE L[] ; int index ;
{

 double dist, line_alpha  ;
 int i=0;
 double px =  p.X ;
 double py = p.Y ;
 double lx1,ly1,lx2,ly2,ltheta, dx,dy;
 int temp =0  ;
 double xstar , ystar , xestar;
 double xtemp,ytemp , A,B;
 while(i < index)
   {
      lx1= L[i].e1.X ; ly1=L[i].e1.Y  ;
      lx2=L[i].e2.X ;  ly2 = L[i].e2.Y ;
      if (lx1<=lx2 && ly1 <= ly2)
```

```
        {
          dx = lx2-lx1;
          dy = ly2 - ly1;
        }
       if(lx2 <= lx1 && ly2 <= ly1)
       {
         dx = lx1 - lx2 ;
         dy = ly1 - ly2 ;
         xtemp = lx1 ;
         ytemp = ly1 ;
         lx1 = lx2;
         ly1 = ly2 ;
         lx2 = xtemp ;
         ly2 = ytemp ;
       }
       if (lx2 <= lx1 && ly1 <= ly2)
       {
         dx = lx2-lx1 ;
         dy = ly2-ly1;
       }
       if (lx1 <= lx2 && ly2 <= ly1)
       {
         dx = lx1 - lx2 ;
         dy = ly1 - ly2 ;
   xtemp = lx1 ;
         ytemp = ly1 ;
         lx1 = lx2 ;
         ly1 = ly2 ;
 lx2 = xtemp ;
         ly2 = ytemp ;
        }

      ltheta = atan2(dy,dx) ;
      A = cos(ltheta) ;
      B = sin(ltheta) ;

      xstar = (px-lx1)* A + (py-ly1) * B;
      ystar = -(px-lx1)* B + (py-ly1) * A ;

      xestar = (lx2-lx1)* A + (ly2-ly1) * B;

      if(xstar < 0 )
```

```
        dist =  sqrt((px-lx1) * (px-lx1) + (py-ly1) *(py-ly1)) ;
     else {if(xstar >= 0 && xstar <= xestar)
         dist = fabs(ystar) ;
     else
     dist =  sqrt((px-lx2) * (px-lx2) + (py-ly2) *(py-ly2)) ;

   }

       if (fabs(dist) < 7)  /* use 7 for delta */
          return 0 ;
       else
            i++ ;
    }
 return 1;
}


/************************************************************
FUNCTION :GetNumLines()
************************************************************/
int GetNumLines()
{
  int  kk ;
  printf("\n ENTER NUM. OF LINES :");
  scanf("%d",&kk);
  return kk ;
}

/**********************************************************
FUNCTION : normalize()
PURPOSE  : return the normalized value of orientation
**********************************************************/

double normalize(double o)
{
double d = o;


  d= o - 2 * PI *(ceil((o+PI)/(2*PI))-1.0);
  return(d);

}
```

```
/********************************/

int sum(int a, int b)
{
  return a+b ;
}


/*********************************************************/
/* FUNCTION : detectEdges()
/* AUTHOR   : KHALED MORSY
/* PURPOSE  : main function for detecting edges by tracking and
/*            random hitting
/*********************************************************/
void detectEdges()
{
  IMG_LINE L, LS[20];
  EDGE Q;
  int yy, hh ;
  int i=0;
  long r0,c;
  int a =1;
  int j = 227;  /* best for 732 by 476 image size */
  int k = 317;
  int num_hits = 0;
  long M,prn;
  POINT p0;
  double p0x,p0y;
   int K;
   prn=0;
  c=j*732+k;
  M=732*476;
  K= GetNumLines();
  while(i<K)
  {
        prn = PseudoRandom(prn,a,c,M);

        p0 = ConvertTo2D(prn) ;

        if(significant(p0))
        {
         if(NotClose(p0,LS,i))
```

```
{
   TrackEdge(p0,Q,&L);
            if (L.n > 50)
             {

                  AddSegment(LS,L,i);
   printf("\nline # %d :%f %f %f %f",
                             i, LS[i].e1.X , LS[i].e1.Y, LS[i].e2.X,
                             LS[i].e2.Y) ;
                  i++ ;
             }
        }
     }
 }
 InitLogLines(NULL,0);
 LogLines(LS,K);

}

/******************************************************/
```

4. IMAGE UNDERSTANDING LIBRARY FUNCTIONS

```
/****************************************************************
   FILE NAME  : imgLib.h
   By         : KHALED MORSY
   Last Update: March 17
   Purpose    : set of functions required by Yamabico IU system
                collected from low-level functions (John Kisor)
                to provide easier user-level interface with the system
****************************************************************/
YAMAIMAGE* b1Image;

/****************************************************************/
/* Function : copyImageToMemory(IMSPage frameNumber)
   Purpose  : Copies an image frame to 2-d array for edge detection
              and further IU algorithms.
   Authors   : John Kisor, Khaled Morsy and L. Remias
****************************************************************/
void copyImageToMemory(IMSPage frameNumber)
{
unsigned long page1Index = 0xfa000000; /* First empty line for pg2 pixels */
unsigned long page2Index = 0xfa07a400; /* First 1K of pixels on page 2*/
unsigned long page1Source = 0xfa000000;
unsigned long page2Source = 0xfa07a400;


int i,j;

int xSize = 732;
int ySize = 476;

  b1Image = (YAMAIMAGE*)malloc(sizeof(YAMAIMAGE));


   b1Image->xSize = xSize;
   b1Image->ySize = ySize;


   for(i=0; i<476; i+=2)
   {
      for (j=0; j<732; j++) /*732 bytes
      {
```

```
          b1Image->image[j][i] = *(BYTE*)page1Source;
          b1Image->image[j][i+1]= *(BYTE*)page2Source;


          page1Source++;
          page2Source++;
            if(j==1&& i==0)  /* for test only */
             printf("\n %d", b1Image->image[j][i]);

     }

     page1Index += 0x400;          /* 1K inc---new row */
     page1Source = page1Index;
     page2Index += 0x400;
     page2Source = page2Index;

   }

}

/*****************************************************************/
/* Function : initImgSys(void)                                   */
/* Purpose  : initalize image grabbing system                    */
/* Author   : Khaled Morsy   March 97                            */
/*****************************************************************/
void initImgSys(void)
{
   DisableInterrupts();
   setInputPath(A1, CONSTANT);
   setInputPath(B1, CONSTANT);
   setInputPath(A0, CONSTANT);
   setFirstKToConstant(A1, 0x0000);
   setFirstKToConstant(B1, 0x0000);
   setFirstKToConstant(B1, 0x0000);

}

/*****************************************************************/
/* Function : setCamera()                                        */
/* Purpose  : set input path to Camera for frames A0,A1,B1       */
/* Author   : Khaled Morsy                                       */
```

```
/****************************************************************/
void setCamera(void)
{
   setInputPath(A1, CAMERA);
   setInputPath(B1, CAMERA);
   setInputPath(A0, CAMERA);
}


/****************************************************************/
/* Function : startGrab()                                       */
/* Purpose  : start grabbing an image to   frames A0,A1,B1      */
/* Author   : Khaled Morsy   March 97                           */
/****************************************************************/
void startGrab(void)
{
  setFrameAcquire(A0, GRAB);
  setFrameAcquire(A1, GRAB);
  setFrameAcquire(B1, GRAB);
}

/****************************************************************/
/* Function : stopGrab()                                        */
/* Purpose  : stop grabbing an image to   frames A0,A1,B1       */
/* Author   : Khaled Morsy   March 97                           */
/****************************************************************/
void stopGrab(void)
{
  setFrameAcquire(A0, FREEZE);
  setFrameAcquire(A1, FREEZE);
  setFrameAcquire(B1, FREEZE);
}


/****************************************************************/
/* Function : snap()                                            */
/* Purpose  : snap an image to            frames A0,A1,B1       */
/* Author   : Khaled Morsy   March 97  (from J. Kisor)          */
/****************************************************************/
void snap(void)
{
  setFrameAcquire(A0, SNAP);
```

```
  setFrameAcquire(A1, SNAP);
  setFrameAcquire(B1, SNAP);
}

/*******************************************************************/
/* Function : clearFrames()                                       */
/* Purpose  : clear pixel data in A0,A1,B1                        */
/* Author   : Khaled Morsy   March 97  (from J. Kisor)            */
/*******************************************************************/
void clearFrames(void)
{
   setInputPath(A1, CONSTANT);
   setInputPath(B1, CONSTANT);
   setInputPath(A0, CONSTANT);
}

/*******************************************************************/
/* Function : interlacePage2(void)
/* Purpose  : Copies the page 2 part of an interlaced image
   between the lines of page one of the interlaced image to make
   a truly interlaced picture
   Author   : John Kisor
   Comments : Khaled Morsy March 97
********************************************************************/

void interlacePage2(void)
{
int i, j;
unsigned long page1Index = 0xfa000000; /* First empty line for pg2 pixels */
unsigned long page2Index = 0xfa07a400; /* First 1K of pixels on page 2*/
unsigned long page1Destination = 0xfa000800;
unsigned long page2Source = 0xfa07a400;

   selectPage(B1);

   for (i=0; i<236; i++) {
      for (j=0; j<366; j++) { /*732 bytes but 2 bytes per access */
         *(WORD*)page1Destination = *(WORD*)page2Source;
         page1Destination += 2;
         page2Source += 2;
      }
      page1Index += 0x800;
```

```
      page2Index += 0x800;
      page1Destination = page1Index;
      page2Source = page2Index;

   }


}

/****************************************************************
 Function : readImage
 Author   : Khaled Morsy
****************************************************************/

void readImage(void)
{
  setCamera();
  startGrab();
  acqEnable();
}
```

APPENDIX B. WORLD MODEL

```
/* FILE:          deck2.h
   By  :          Khaled Morsy
   Notes:         2D description + height of the 2nd floor points.
  Changes:        Major change from the 3D model of 5th floor by J. Stein.
                  Only, z-value added to a vertx. All other
                  functions are deleted.
*********************************************************************/
 WORLD *make_world()
{
  WORLD *W;
  POLYGON *P1,*Last_p;

  VERTEX *P1V1, *P1V2, *P1V3, *P1V4, *P1V5, *P1V6, *P1V7,
         *P1V8,*P1V9,*P1V10, *P1V11, *P1V12, *P1V13, *P1V14,
         *P1V15,*P1V16,*P1V17, *P1V18, *P1V19, *P1V20, *P1V21,
         *P1V22,*P1V23,*P1V24, *P1V25, *P1V26, *P1V27, *P1V28,
         *P1V29,*P1V30,*P1V31, *P1V32, *P1V33, *P1V34, *P1V35,
         *P1V36,*P1V37,*P1V38, *P1V39, *P1V40, *P1V41, *P1V42,
         *P1V43,*P1V44,*P1V45, *P1V46, *P1V47, *P1V48, *P1V49,
         *P1V50,*P1V51,*P1V52, *P1V53, *P1V54, *P1V55, *P1V56,
         *P1V57,*P1V58,*P1V59, *P1V60, *P1V61, *P1V62, *P1V63,
         *P1V64,*P1V65,*P1V66, *P1V67, *P1V68, *P1V69, *P1V70,
         *P1V71,*P1V72,*P1V73, *P1V74, *P1V75, *P1V76, *P1V77,
         *P1V78,*P1V79, *P1V80, *P1V81, *P1V82,*P1V83, *P1V84,
         *P1V85,*P1V86, *P1V87, *P1V88, *P1V89, *P1V90,
         *P1V91,*P1V92,*P1V93, *P1V94, *P1V95, *P1V96, *P1V97,
         *P1V98,*P1V99, *P1V100, *P1V101, *P1V102, *P1V103,
         *P1V104,*P1V105,*P1V106, *P1V107, *P1V108, *P1V109,*P1V110,
         *P1V111,*P1V112,*P1V113, *P1V114, *P1V115, *P1V116,
         *P1V117,*P1V118,*P1V119,*P1V120, *P1V121, *P1V122,*P1V123,
         *P1V124,*P1V125,
         *P1V126,*P1V127,*P1V128, *P1V129, *P1V130, *P1V131,
         *P1V132,*P1V133,*P1V134, *P1V135, *P1V136, *P1V137,
         *P1V138;

  W=add_world("2nd_floor",9);
  H1=add_ph("front_hall",10,W,1,0);
  P1=add_pg(W,1);    /* CW Polygon */
```

```
P1V1  = add_vertex(P1,0.0,0.0,257.5);         /* beside graphics lab */
P1V2  = add_vertex(P1,0.0,248.5,257.5);

P1V3  = add_vertex(P1,164.7,248.5,212.5);     /*rm 253A */
P1V4  = add_vertex(P1,164.7,256.75,212.5);
P1V5  = add_vertex(P1,245.7,256.75,212.5);
P1V6  = add_vertex(P1,245.7,248.5,212.5);

P1V7  = add_vertex(P1,390.2,248.5,212.5);     /* EXIT DOOR */
P1V8  = add_vertex(P1,390.2,256.75,212.5);
P1V9  = add_vertex(P1,491.2,256.75,212.5);
P1V10 = add_vertex(P1,491.2,248.5,212.5);

P1V11 = add_vertex(P1,657.8,258.5,283.5);
P1V12 = add_vertex(P1,657.8,845.5,283.5);
P1V13 = add_vertex(P1,893.5,845.5,283.5);

P1V14 = add_vertex(P1,893.5,771.9,212.5);     /* elev 1 */
P1V15 = add_vertex(P1,956.5,771.9,212.5);
P1V16 = add_vertex(P1,956.5,818.9,212.5);
P1V17 = add_vertex(P1,1096.68,818.9,212.5);
P1V18 = add_vertex(P1,1096.68,588.9,212.5);
P1V19 = add_vertex(P1,956.5,588.9,212.5);
P1V20 = add_vertex(P1,956.5,635.9,212.5);
P1V21 = add_vertex(P1,893.5,635.9,212.5);

P1V22 = add_vertex(P1,893.5,477.4,212.5);     /* elev 2 */
P1V23 = add_vertex(P1,956.5,477.4,212.5);
P1V24 = add_vertex(P1,956.5,524.7,212.5);
P1V25 = add_vertex(P1,1096.68,524.7,212.5);
P1V26 = add_vertex(P1,1096.68,294.7,212.5);
P1V27 = add_vertex(P1,956.5,294.7,212.5);
P1V28 = add_vertex(P1,956.5,341.7,212.5);
P1V29 = add_vertex(P1,893.5,341.7,212.5);
P1V30 = add_vertex(P1,893.5,248.5,212.5);     /* end of elevators region (corner)*/

P1V31 = add_vertex(P1,1190.5,248.5,212.5);    /* room 261 */
P1V32 = add_vertex(P1,1190.5,256.75,212.5);
P1V33 = add_vertex(P1,1281.5,256.75,212.5);
P1V34 = add_vertex(P1,1281.5,248.5,212.5);

P1V35 = add_vertex(P1,1592.5,248.5,212.5);    /* room 259 */
```

```
P1V36 = add_vertex(P1,1592.5,256.75,212.5);
P1V37 = add_vertex(P1,1683.9,256.75,212.5);
P1V38 = add_vertex(P1,1683.9,248.5,212.5);

P1V39 = add_vertex(P1,2159.3,248.5,212.5) ;  /* room 257 */
P1V40 = add_vertex(P1,2159.3,256.75,212.5);
P1V41 = add_vertex(P1,2250.3,256.75,212.5);
P1V42 = add_vertex(P1,2250.3, 248.5,212.5);

P1V43 = add_vertex(P1,2587.3,248.5,212.5);   /* room 243 */
P1V44 = add_vertex(P1,2587.3,256.75,212.5);
P1V45 = add_vertex(P1,2678.3,256.75,212.5);
P1V46 = add_vertex(P1,2678.3,248.5,212.5);

P1V47 = add_vertex(P1,3692.0, 248.5,212.5);  /* corner */

P1V48 = add_vertex(P1,3692.0,457.0,212.5);   /* room 241 */
P1V49 = add_vertex(P1,3683.75,457.0,212.5);
P1V50 = add_vertex(P1,3683.75,548.0,212.5);
P1V51 = add_vertex(P1,3692.0,548.0,212.5);

P1V52 = add_vertex(P1,3692.0,212.5);         /* corners */
P1V53 = add_vertex(P1,3711.6,727.3,283.5);
P1V54 = add_vertex(P1,3711.6,809.7,283.5);
P1V55 = add_vertex(P1,3962.2,809.7,283.5);

P1V56 = add_vertex(P1,3962.2,212.5);         /* room 239 */
P1V57 = add_vertex(P1,3970.45,358.9,212.5);
P1V58 = add_vertex(P1,3970.45,267.9,212.5);
P1V59 = add_vertex(P1,3962.2,267.9,212.5);

P1V60 = add_vertex(P1,3962.2,248.5);          /* corner */

P1V61 = add_vertex(P1,5269.7,248.5,212.5);    /* 235 */
P1V62 = add_vertex(P1,5269.7,256.75,212.5);
P1V63 = add_vertex(P1,5360.7,256.75,212.5);
P1V64 = add_vertex(P1,5360.7,248.5,212.5);

P1V65 = add_vertex(P1,5680.7,248.5,212.5);    /* front of 231 */
P1V66 = add_vertex(P1,5680.7,375.0,212.5);
P1V67 = add_vertex(P1,5819.2,375.0,212.5);
P1V68 = add_vertex(P1,5819.2,248.5,212.5);
```

```
P1V69 = add_vertex(P1,6239.2,248.5,257.5);   /* end of hallway */
P1V70 = add_vertex(P1,6239.2,0.0,257.5);     /* end of hallway */

P1V71 = add_vertex(P1,5932.7,0.0,212.5);     /* rm 228 */
P1V72 = add_vertex(P1,5932.7,-8.25,212.5);
P1V73 = add_vertex(P1,5841.7,-8.25,212.5);
P1V74 = add_vertex(P1,5841.7,0.0,212.5);

P1V75 = add_vertex(P1,5364.2,0.0,212.5);      /* rm 226 */
P1V76 = add_vertex(P1,5364.2,-8.25,212.5);
P1V77 = add_vertex(P1,5273.2,-8.25,212.5);
P1V78 = add_vertex(P1,5273.2,0.0,212.5);

P1V79 = add_vertex(P1,4976.0,0.0,212.5);     /* rm 230 */
P1V80 = add_vertex(P1,4976.0,-8.25,212.5);
P1V81 = add_vertex(P1,4884.2,-8.25,212.5);
P1V82 = add_vertex(P1,4884.2,0.0,212.5);

P1V83 = add_vertex(P1,4796.2,0.0,212.5);     /* 232 */
P1V84 = add_vertex(P1,4796.2,-8.25,212.5);
P1V85 = add_vertex(P1,4715.7,-8.25,212.5);
P1V86= add_vertex(P1,4715.7 ,0.0,212.5);

P1V87 = add_vertex(P1,4534.0, 0.0,212.5);    /*  around  water cooler */
P1V88 = add_vertex(P1,4534.0, -48.0,212.5);
P1V89 = add_vertex(P1,4447.9, -48.0,212.5);
P1V90 = add_vertex(P1,4447.9, 0.0,212.5);

P1V91 = add_vertex(P1,4367.9, 0.0,212.5);    /* 234  */
P1V92 = add_vertex(P1,4367.9, -8.25,212.5);
P1V93 = add_vertex(P1,4287.4 ,-8.25,212.5);
P1V94 = add_vertex(P1,4287.4, 0.0,212.5);

P1V95 = add_vertex(P1,4228.4, 0.0,212.5);    /* EXIT B*/
P1V96 = add_vertex(P1,4228.4, -8.25,212.5);
P1V97 = add_vertex(P1,4046.4, -8.25,212.5);
P1V98 = add_vertex(P1,4046.4, 0.0,212.5);

P1V99  = add_vertex(P1,3999.8, 0.0,212.5);    /* 236 */
P1V100 = add_vertex(P1,3999.8, -8.25,212.5);
P1V101 = add_vertex(P1,3919.3, -8.25,212.5);
```

```
P1V102 = add_vertex(P1,3919.3, 0.0,212.5);

P1V103 = add_vertex(P1,3351.3, 0.0,212.5);   /* rm 238 */
P1V104 = add_vertex(P1,3351.3, -18.25,212.5);
P1V105 = add_vertex(P1,3260.3, -18.25,212.5);
P1V106 = add_vertex(P1,3260.3, 0.0 ,212.5);

P1V107 = add_vertex(P1,3018.8, 0.0,212.5);   /* 240  maze room*/
P1V108 = add_vertex(P1,3018.8, -8.5,212.5);
P1V109 = add_vertex(P1,2927.0, -8.5,212.5);
P1V110 = add_vertex(P1,2927.0,0.0,212.5);

P1V111 = add_vertex(P1,2123.0,0.0,212.5);    /* 242  yamabico lab */
P1V112 = add_vertex(P1,2123.0,-8.5,212.5);
P1V113 = add_vertex(P1,2032,-8.5,212.5);
P1V114 = add_vertex(P1,2032,0.0,212.5);

P1V115 = add_vertex(P1,1971.4, 0.0,212.5);   /* rm 244 */
P1V116 = add_vertex(P1,1971.4, -8.25,212.5);
P1V117 = add_vertex(P1,1880.4, -8.25,212.5);
P1V118 = add_vertex(P1,1880.4, 0.0,212.5);

P1V119 = add_vertex(P1,1553.2, 0.0,212.5);   /* rm 246 */
P1V120 = add_vertex(P1,1553.2, -8.25,212.5);
P1V121 = add_vertex(P1,1462.2, -8.25,212.5);
P1V122 = add_vertex(P1,1462.2, 0.0,212.5);

P1V123 = add_vertex(P1,1311.9, 0.0,212.5);   /* rm 248 */
P1V124 = add_vertex(P1,1311.9,-8.25,212.5);
P1V125 = add_vertex(P1,1220.9,-8.25,212.5);
P1V126 = add_vertex(P1,1220.9,0,212.5);

P1V127 = add_vertex(P1,834.4,0.0,212.5);     /* rm 250 */
P1V128 = add_vertex(P1,834.4,-8.25,212.5);
P1V129 = add_vertex(P1,743.4,-8.25,212.5);
P1V130 = add_vertex(P1,743.4,0.0,212.5);

P1V131 = add_vertex(P1,417.1,0.0,212.5);     /* rm 252 */
P1V132 = add_vertex(P1,417.1,-8.25,212.5);
P1V133 = add_vertex(P1,326.1,-8.25,212.5);
P1V134 = add_vertex(P1,326.1,0.0,212.5);
```

```
P1V135 = add_vertex(P1,265.9,0.0,212.5);     /* rm 254 */
P1V136 = add_vertex(P1,265.9,-8.25,212.5);
P1V137 = add_vertex(P1,174.9,-8.25,212.5);
P1V138 = add_vertex(P1,174.9,0.0,212.5);
/*** end of floor points */
```

LIST OF REFERENCES

[1] J. R. Parker, *Practical Computer Vision Using C*, John Wiley and Sons, 1994.

[2] L. J. Galbiati, *Machine Vision and Digital Image Processing Fundamentals*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1990.

[3] D. H. Ballard and C.M. Brown, *Computer Vision*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1982.

[4] A. K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1989.

[5] E. R. Davies, *Machine Vision: Theory, Algorithms, Practicalities* Academic Press, London, 1990.

[6] S. W. Zuker, Region growing: childhood and adolescence. Computer Graphics and Image processing, 5, pages 382-399.

[7] C. E. Thorpe, *Vision and Navigation*, KAP, 1990.

[8] I. J. Cox and G. T. Wilfong, *Autonomous Robot Vehicles*, Springer-Verlag, page xix, 1990.

[9] J. G. Kovalchik, *Layered Motion Planning for Autonomous Mobile Robots using Free Space Decomposition and Steering Functions.* Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, 1995.

[10] C. L. Chuang, *Layered Safe Motion Planning for Autonomous Vehicles.* Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, 1995.

[11] M. A. Wahdan, *New Motion planning and Real-Time Localization Methods Using Proximity for Autonomous Mobile Robots.* Ph.D. dissertation, Naval Postgraduate School, Monterey, CA, 1996.

[12] X. Yun and K. C. Tan, A wall-following method for escaping local minima in potential field based motion planning. *Proc. 8th International Conference on Advanced Robotics*, pages 421–426, July 1997.

[13] A. Kosaka and A. C. Kak, Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties, *CVGIP: Image Understanding*, Vol. 56, No. 3, pages 271-329, Nov. 1992.

[14] M. Bishay, R. A. Peters II, and K. Kawamura, Generation of architectural CAD models using a mobile robot, *Proc. IEEE Int. Conf. on Robotics & Automat.*, pages 711–717, 1994.

[15] W. L. Xu, S. K. Tso, and Y .H. Fung, Sensor-based reactive Navigation of a mobile robot through local target switching. *Proc. 8th International Conference on Advanced Robotics*, pages 361–366, July 1997.

[16] M. E. Cleary and J. D. Crisman, User instruction for semi-autonomous mobile robot control. *Proc. 8th International Conference on Advanced Robotics*, pages 379–384, July 1997.

[17] Y. Kanayama. Two dimensional wheeled vehicle kinematics. *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3079–3084, May 1994.

[18] Y. Kanayama and T. Noguchi. Spatial learning by an autonomous mobile robot with ultrasonic sensors. *University of California Santa Barbara Dept. of Comp. Science. Technical Report TRCS89-06*, February 1989.

[19] S. R. Sherfy. *A mobile Root sonar system*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1991.

[20] J. R. Millan and A, Arleo, Neural network learning of variable grid-based maps for the autonomous navigation of robots, *Proc. IEEE International Symposium on Computational Intellignce in Robotics and Automation*, pages 40–45, July 1997.

[21] E. Tunstel, H. Danny, T. Lippincott, and M. Jamshidi, Autonomous navigation using an adaptive hierarchy of multiple fuzzy-behaviors, *Proc. IEEE International Symposium on Computational Intellignce in Robotics and Automation*, pages 276–281, July 1997.

[22] J. C. Latombe, *Robot Motion Planning*, KAP, 1990.

[23] X. Lebegue and J.K. Aggarwal. Generation of architectural CAD models using a mobile robot. *Proc. IEEE Int. Conf. on Robotics & Automat.*, pages 711–717, 1994.

[24] W. Burger and B. Bhanu. A geometric constraint method for estimating 3-d camera motion. *Proc. IEEE Int. Conf. on Robotics & Automat.*, pages 1155–1160, 1994.

[25] C. J. Taylor and D. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. In Algorithmic Foundation Of Robotics, pages 69–83, 1995.

[26] D. P. Huttenlocher, M. E. Leventon and W. J. Rucklidge. Visually-guided navigation by comparing edge images. In Algorithmic Foundation of Robotics, pages 85–96, 1995.

[27] A. Lazanas and J. C. Latombe. Landmark-based robot navigation. *Proc. American Assoc. Artificial Intelligence,* July 1992.

[28] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. *Proc. 32nd Annual Symposium on Foundations of Computer Science,* page 298, October 1991.

[29] T. Shibata, Y. Matsumoto, and T. Kuwahara. Hyper Scooter: a mobile robot sharing visual information with a human *Proc. IEEE Int. Conf. on robotics and automation*, pages 1074-1079, 1994.

[30] H. Mori, N. M. Charkari, and S. Kotani. Danger estimation of vehicles at intersection. *Proc. of the IEEE Conf. on Robotics and Automation*, pages 781–787, 1995.

[31] A. Kosaka, M. Meng, and A. C. Kak. Vision-guided mobile robot navigation using retroactive updating of position uncertainties *Proc. of the IEEE Conf. on Robotics and Automation*, pages 1–7, 1993.

[32] B. Rochwerger, C. L. Fennema, B. Draper, A. R. Hanson, and E. M. Riseman. Executive reactive behavior for autonomous navigation. *In CVPR.* Seattle, WA, pages 838–841, 1994.

[33] D. L. Macpherson, *Automated Cartography by an Autonomous Mobile Robot using Ultrasonic Range Finders*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, 1993.

[34] J. T. Lochner, *Analysis and Improvement of an Ultrasonic Sonar System on an Autonomous Mobile Robot*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1994.

[35] K. R. Peterson, *Visual Navigation For an Autonomous Mobile Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1992.

[36] J. E. Stein, *Modeling, Visibility Testing and Projection of an Orthogonal Three Dimensional World in Support of a Single Camera System,* Master's Thesis, Naval Postgraduate School, Monterey, CA, 1992.

[37] M. J. DeClue, *Object Recognition Through Image Understanding for an Autonmous Mobile Robot*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1993.

[38] J. C. Kisor, *Integration of an Image Hardware/Software System into an Autonomous Robot,* Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.

[39] R. Bolles, Verification vision for programmable assembly, *Proc. 5th IJCAI*, 1977.

[40] P. G. Selfridge, J.M. Prewitt, C.R. Dyer, and S. Ranade, Segmentation algorithms for abdominal computerized tomography scans, *Proc. 3rd COMPSAC*, pages 571–577, November 1979.

[41] R. O. Duda and P.E. Hart, Use of the Hough transformation to detect lines and curves in pictures, *Commun. ACM15, 1*, 1972.

[42] S. A. Dudani and A. L. Luck. Locating straight-line edge segments on outdoor scenes. *Pattern Recognition 10*, pages 145-157, 1978.

[43] A. Martilli. Edge detection using heuristic search methods. *CGIP 1,2*, August 1972.

[44] N. J. Nilson. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[45] N. J. Nilson. *Principles of Artificial Intelligence.* Tioga Press, Palo Alto, CA, 1980.

[46] H. J. Barrow. Interactive aids for cartography and photo interpretation. *Semi-Annual Technical Report, AI center, SRI international*, 1976.

[47] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Academic Press, New York, 1976.

[48] A. Martilli. An application of heuristic search methods to edge and contour detection. *Commun. ACM 19,2*, 1976.

[49] J. B. Burns, A. R. Hanson, and E. M. Reseman. Extracting straight lines. *IEEE Trans. Pattern Anal. Machine Intell., PAMI-8*, pages 425-455, July 1986.

[50] P. Kahn, L. Kitchen, and E. M. Riseman. A fast line finder for vision-guided robot navigation. *IEEE Trans. Pattern Anal. Machine Intell., PAMI-12*, pages 1098-1102, November 1990.

[51] M. K. Leung, Y. Liu and Thomas S. Huang. Estimating 3D vehicle motion in an outdoor scene from monocular and stereo image sequences. *IEEE workshop on Visual Motion*, pages 62-67, New Jersey, 1991.

[52] V. Venkateswar. Extraction of straight lines in aerial images. *IEEE Trans. Pattern Anal. Machine Intell., PAMI-14*, pages 1111-1114, November 1992.

[53] R. C. Nelson. Finding line segments by stick growing. *IEEE Trans. Pattern Anal. Machine Intell., PAMI-16*, pages 519-523, May 1994.

[54] A. Nowak, A. Florek, T. Piascik. Edge tracing in a priori known direction. *Proc. European Conference on Computer Vision*, pages 38-42, 1992.

[55] D. Marr and E.Hildreth, Theory of edge detection, *Proc. Royal Soc. London*, B-207, pages 187-217, 1980.

[56] J. Canny, A computational approach to edge detection, *IEEE Trans. PAMI-8*, Number 6, November 1986.

[57] S. Sarkar and K. L. Boyer, On optimal infinite impulse response edge detection, *IEEE Trans. PAMI-13*, Number 11, November 1991.

[58] E. R. Davies and A. I. C. Johnstone, Engineering trade-offs in the design of a real-time system of the visual inspection of small products, *Proc. 4th Conference on UK Research in Advanced Manufacture*, IMechE Conference Publication, London, pages 15–22, December 1986.

[59] E. R. Davies and A. I. C. Johnstone, Methodology for optimizing cost/speed tradeoffs in real-time inspection hardware, *IEE Proc. E 136*, pages 62–69, 1998.

[60] J. L.Crowely. World modeling and position estimation for a mobile robot using ultrasonic ranging. *Proc. IEEE International Conference on Robotics and Automation*, page 674–680, 1989.

[61] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, vol. 24, pages 381–395, 1981.

[62] S. Linnainmaa, D. Harwood, and L. S. Davis. Pose determination of a three-dimensional object using triangle pairs. *IEEE Trans. Pattern Anal. Machine Intell., PAMI-10*, Num. 5, pages 634-647, 1988.

[63] R. Kumar. Determination of camera location and orientation. *Proc. 1989 DARPA Image Understanding Workshop*, pages 870–879, Morgan Kufmann, San Mateo, California, 1989.

[64] K. Sugihara. Some location problems for robot navigation using a single camera, *Comp. vision, graph., and image proc., 42* , page 112-129, 1988.

[65] C. Pegard and El Mustapha Mouaddib. A mobile robot using a panoramic view *Proc. IEEE International Conference on Robotics and Automation*, pages 89–94, 1996.

[66] G. Dudek and Z. Chang. Vision-based localization without explicit object models. *Proc. IEEE International Conference on Robotics and Automation*, pages 76–82, 1996.

[67] I. Pitas. *Digital Image Processing Algorithms.* Prentice-Hall Inc., Englewood Cliff, NJ, 1995.

[68] Y. Kanayama. Introduction to theoretical robotics. *Lecture Notes of the Advanced Robotics Course.* Department of Computer Science, Naval Postgraduate School, 1996.

[69] Y. Kanayama and S. Yuta. A Sonar range finding module for mobile robots. *Proc. 14 International Symposium on Industrial Robots*, pages 643-652, 1984.

[70] K. A. Morsy and Y. Kanayama. A new straight edge detection method using direction-controlled edge tracking and random hitting. *Proc. The 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA, July 1997.

[71] D. E. Knuth. *The Art of Computer Programming Vol II*, Addison-Wesley, Reading, MA, 1973.

[72] Y. Kanayama, D. MacPherson, and G. Krahn. Two dimensional transformations and Its application to vehicle motion control and analysis, *IEEE International Conference on Robotics and Automation*, pages 13-18, 1993.

[73] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes. *Computer Graphics Principles and Practice,* Addison-Wesley, Reading, MA, 1993.

[74] R. Fosner. *OpenGL Programming for Windows95 and Windows NT* Addison-Wesley, Reading, MA, 1997.

[75] J. Vince. *3-D Computer Animation* Addison-Wesley, Reading, MA, 1992.

[76] Y. Kanayama, K. Kimura, F. Miyazaki, and T. Noguchi. A stable tracking control method for an autonomous mobile robot. *IEEE International Conference on Robotics and Automation*, pages 1315–1317, 1988.

[77] Y. Kanayama and M. Onishi. Locomotion functions for a mobile robot language, mml. *IEEE International Conference on Robotics and Automation*, pages 1110–1115, 1991.

[78] Ironics, Inc., *IV-SPARC-25A/33A VMEbus Single Board Super Computer and MultiProcessing Engine—User's Manual.*

[79] Imaging Technology Inc, IMS Hardware Ref. Manual, Technical Reference, Apr. 1993.

[80] Cohu, Inc., *Installation and Operation Manual for 4110 Digital Output Monochrome CCD camera,* Technical Manual Code 6X-899, August 1990.

[81] Lynne Grewe and R. R. Brooks, On localization of objects in the wavelet domain, *Proc. The 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA, July 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road., Ste 0922
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School 2
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS
   Department of Computer Science 1
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Professor Yutaka Kanayama, Code CS/Ka 2
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5101

5. Professor C. Thomas Wu, Code CS/Wq 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943-5101

6. Professor Craig Rasmussen, Code MA/Ra 1
   Department of Mathematics
   Naval Postgraduate School
   Monterey, CA 93943-5101

7. Professor Xiaoping Yun, Code ECE/Yu 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5101

8. Professor Lynne L. Grewe.................................. 1
   CSUMB
   100 Campus Center
   Seaside, CA 939551

9. Egyptian Military Attache 1
   2308 Tracy Place NW
   Washington, DC 20008

10. Egyptian Armament Authority - Training Department 1
c/o American Embassy (Cairo, Egypt)
Office of Military Cooperation
Box 29 (TNG)
FPO, NY 09527-0051

11. Military Technical College (Egypt) 1
c/o American Embassy (Cairo, Egypt)
Office of Military Cooperation
Box 29 (TNG)
FPO, NY 09527-0051

12. Military Research Center (Egypt) 1
c/o American Embassy (Cairo, Egypt)
Office of Military Cooperation
Box 29 (TNG)
FPO, NY 09527-00511

13. Dr. Mahmoud Wahdan 1
5 El-Shrif Street
Roxy Cairo
Egypt

14. LTC. Hesham Eldeeb, Code ECE/Ph 2
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5101

15. Maj. Khaled Ahmed Morsy 3
30 Manshyet Elbakry St, Apt. 21
Heliopolies, Cairo, Egypt

16. Maj. Ashraf Mamdouh, Code ECE/Ph 2
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5101